TimLarson

Any help on these projects is welcome (comments, use cases, design, implementation, bugspray, etc.). You can contact me at tim at keow dot org or via the Cocoon dev or users list or the xReporter list.

(I am retiring the old address, timlarsonwork at yahoo dot com)

Current Project(s)

- Form and report design GUI
  - Initially for xReporter] and [http://xreporter.cocoondev.org, but hopefully others will find additional uses.
  - Involved creating the EffectTransformer. This allows the structure of the source code of Java XML transformers to match the structure of the data being transformed.
  - Required adding support to Cocoon Forms for dynamic creation and deletion of widgets approximately as described in this email. See below for the results thus far.

Additional topics/links:

New widget definition builder design:
Email Link

Another short writeup about cforms builder logic pools:

Setup the builder's initial context.

Create a form by passing the Assistant

to the form definition class's constructor.

The form definition then asks the Assistant

for everything it needs.

The Assistant then returns the data from its

cache, queries the appropriate builder for the

data, or queries the appropriate builder logic pool.

This allows most definitions to not require specific

builder classes to be created, as the general logic to

retrieve the configuration data that they need is in the

builder logic pool.

New design for dynamic widgets, widget definition repositories, etc:
WoodyScratchpad

Design for autogeneration of bindings and templates:
Email Link (skip down to the part that starts: "Allow model, binding, and template fragments to be associated with each other")

New features for Cocoon Forms (Woody):

Update

The features described below are present in Cocoon version 2.1.4, but they are being redesigned at WoodyScratchpad. Please join the discussion to make sure these features get properly designed meet your needs.

Class, New, Struct, and Union

Quick summary

Combined use of these new features allow for the creation of dynamic, recursive GUI's and editors for structured content of any nesting level. See "TODO" below for current limitations in this implementation. When finished, we should be able to use Cocoon Forms to create and edit sitemaps, form definitions and templates, xreporter reports, etc.

Class

- Definition element used to create a reusable collection of widget definitions.
- Template element used to create a reusable collection of templates and/or markup.

New

- Definition element used to insert the widgets from a referenced class.
- Template element used to insert the templates and/or markup from a referenced class.

TODO: Should describe plans for class instantiation parameters, etc. here.

Struct

Container widget which can hold zero or more widgets.

This wraps the set of contained widgets in their own namespace (the id of the Struct widget), allowing the same widget id's to be reused multiple times in a form. Also, using a Struct widget as a Union case allows the case to contain multiple widgets.

TODO: Should describe the use of the corresponding template element here.

Union

Discriminated union widget which holds one of several possible widgets at any one time.

TODO: Should describe the use of the corresponding template elements here.

Development Notes: Watch for changes coming to this definition to experiment with various uses for this widget type, such as for the definition of subforms.

Probably going to change this from a CompositeWidget to a ContainerWidget and allow the union case to be specified by a referenced widget. This may involve allowing the referenced widget to derive its value from a general expression referencing other widgets. There is some design work left to figure out how to get deterministic behaviour from derived widgets which reference other derived widgets (order of evaluation issues).
[This change has now happened, and is reflected in the detailed notes below.]

Detailed descriptions:

Class

Specify a set of widget definitions or templates and/or markup for reuse. The same concept is implemented in the form definition for widget reuse, and in the form template for template and/or markup reuse.

Definition syntax

Only allowed as a direct child widget of wd:form. Note that the definition class id's currently form a flat namespace.

```
<wd:class id="some-class">
  <wd:widgets>
    <!-- A list of any widgets(s) (e.g. wd:field, wd:new, wd:struct, wd:union) -->
    [...]
  </wd:widgets>
</wd:class>
```

Template syntax

Inside the wd:form template, allowed anywhere other templates are allowed Note that the template class id's currently form a flat namespace

```
<wt:class id="some-class">
  <!--
    A list of templates and/or markup
  -->
</wt:class>
```

New

Insert the widgets or templates and/or markup from the referenced class.

Definition syntax

New inserts the widgets from the referenced class.

```
<wd:new id="some-class"/>
```

Template syntax

New inserts the templates and markup from the referenced class.

```
<wt:new id="some-class"/>
```

Struct

Generic container for widgets. Useful to hold multiple widgets in a union case, and anywhere there is a need to wrap a collection of widgets in a namespace.

Definition syntax

```
<wd:struct id="some-struct">
  <wd:label>...</wd:label>
  <wd:hint>...</wd:hint>
  <wd:help>...</wd:help>
  <wd:widgets>
    <!--
      A list of any widget(s) (e.g. wd:field, wd:new, wd:struct, wd:union)
    -->
    [...]
  </wd:widgets>
</wd:struct>
```

Template syntax

```
<wt:struct id="some-struct">
  <!--
    A list of templates for contained widgets, possibly mixed with markup
  -->
</wt:struct>
```

Union

Discriminated union. Each direct child widget is considered a union case. The widgets are only created when their case is selected (lazy construction), but their values continue to exist after switching to another case, to allow for case-switching validation, value copying between cases, and automatic support for remembering values on a switch back to a previously selected case. A container widget (Repeater, Struct, Union, or sort-of AggregateField) must be used for union cases that need to hold multiple widgets. The "case" attribute of the union element references the widget that determines the current case.

Definition syntax

```
<wd:union id="widget-id" case="widget-id">
  <wd:label>...</wd:label>
  <wd:hint>...</wd:hint>
  <wd:help>...</wd:help>
  <wd:datatype base="...">
    [...]
  </wd:datatype>
  <wd:selection-list .../>
  <wd:on-value-changed>
    ...
  </wd:on-value-changed>
  <wd:widgets>
    <!--
      A list of any widget(s) (e.g. wd:field, wd:new, wd:struct, wd:union)
    -->
    [...]
  </wd:widgets>
</wd:union>
```

Template syntax

The wt:union template outputs any markup outside of the cases and the contents of the wt:case template for the current union case, while suppressing output from the other wt:case templates.

```
<wt:union>

  <!-- A list of union cases, possibly mixed with markup. -->

  <!-- Use an empty id to handle the case of an empty union discriminant. -->
  <wt:case id="">
    <!-- Any markup -->
  </wt:case>

  <!-- Use the union child widget id as the case id. -->
  <wt:case id="some-case">
    <!--
      A list of any relevant templates, possibly mixed with markup.
    -->
  </wt:case>

</wt:union>
```

Binding syntax

The wb:union binding wraps wb:case bindings. The wb:case binding that matches the current case of the surrounding union is the only wb:case binding which will perform its child bindings.

```
<wb:union>

  <wb:case id="">
    <!-- A list of bindings relevant to the default case -->
  </wb:case>

  <wb:case id="some-case">
    <!-- A list of bindings relevant to the selected case -->
  </wb:case>

</wb:union>
```

TODO:

- Fix issues with nested Repeater's:
  - Forgetting values on Union case switch. (broken in 2.1.5, fixed in CVS)
- Finish "Form GUI" sample:
  - Correct and finish the save side of the binding to allow saving the edited form.
  - Add the rest of the widgets and options.
  - Add display of form as it is created, possibly using frames.
  - Replace the XSP success page with a JX page.
  - Create samples for a form template GUI, form binding GUI, etc. and combine them into a complete form creation and editing GUI.
  - Get somebody with HTML and CSS knowledge to smooth out the visual design.
- Add automatic selection list generation to the union widget definition.
- Add a test to NewDefinition.resolve for union defaults that would case a non-terminating recursion.
- Make the non-terminating recursion detection more efficient.
- Verify the most useful and correct source code locations are given in exceptions.
- Augment the EffectTransformer/WoodyTemplateTransformer exceptions with source code location information.
- Figure out what wi:* element we want to generate for wt:struct and wt:union.
- Make remaining exceptions use I18n catalogue.
- Reduce object turnover in EffectTransformer and WoodyTemplateTransformer. (Should be pretty simple; replace some of the instances of out.copy() with a version out.startElement() or out.endElement() that does not create objects on the element stack, etc.)
- Consider whether widget definition classes and template and/or markup classes should stay in flat namespaces or allow nesting class definitions. (Note that class instances (wd:new, wt:new) can already nest.)
- Move definition and template classes into their own separate namespace.
- Find hotspots in the code to optimize. Currently gets very slow at about seven nesting levels. (Solved by Bruno, already mostly fixed in CVS, just one more case to handle)
- Improve the javadocs.
- Write lots of test cases.
- And hopefully much more...