WoodyFileFormats

Work in progress !

The long discussion about Woody led to restructure its current file formats and content (as of 2003-08-01)

The following files have been identified :

- format dictionary (new), containing converter declarations for the base types declared in cocoon.xconf
- data dictionary (new), containing only datatypes deriving from the base types declared in cocoon.xconf
- form definition, defining widgets
- form template, instanciating widget definitions in a page layout

Previously, one namespace was used for one file, but the discussion showed that namespaces actually represent the different Woody concerns, and that each file contained elements of several concerns and as such could mix namespaces.

The namespaces currently identified are :

- Definition : http://apache.org/cocoon/woody/definition/1.0
- Template : http://apache.org/cocoon/woody/template/1.0
- Instance : http://apache.org/cocoon/woody/instance/1.0
- Binding : http://apache.org/cocoon/woody/binding/1.0

Format dictionary

The format dictionary defines reusable, preconfigured formats that can be referenced both by the data dictionary and form definition.

```
<wd:formats xmlns:wd="http://apache.org/cocoon/woody/definition/1.0">

  <wd:format id="xml-date" basetype="date" converter="formatting">
    <!-- converter-specific configuration -->
    <patterns>
      <pattern>yyyy-MM-dd</pattern>
    </patterns>
  </wd:format>

  <!-- "converter" unspecified, so default converter for "decimal" is used -->
  <wd:format id="price" basetype="decimal" variant="number">
    <patterns>
      <pattern>#.00</wd:pattern>
    </patterns>
  </wd:format>

</wd:formats>
```

The `wd:format` element accepts the following attributes :

- `id` : the format identifier, must be unique
- `basetype` : the base type on which the format applies. This is used to get the appropriate converter.
- `converter` : the converter to use. If not specified, the default converter for the base type is used.
- any other converter-specific attribute, e.g. `variant` in the above example.

_Open items :

- I changed `wd:convertor` to `wd:format`, as a "converter" can convert from a class to another while "format" is more explicit about the fact that one the two classes is `String` (see also java.text.Format)
- The contents of `wd:format` being converter-specific, it's not in the "wd:" namespace (mimics the configurations in `map:components`)
- For the particular FormattingDateConvertor and FormattingDecimalConvertor, verbosity could be reduced by removing the `<patterns>` container element since the configuration doesn't contain anything else than a list of `<pattern>`._

Datatype dictionary

The datatype dictionary contains reusable datatypes that can be used by the form definition.

Datatypes can use an existing format defined in the format dictionary, or define inline anonymous formats. If a datatype doesn't specify a format, the default format for that type, as defined in the xconf file is used.

Datatypes can have validation rules that restrict the set of allowed values compared to that of the base type.

Datatypes can have selection lists that restrain the range of possible values.

```
<wd:datatypes xmlns:wd="http://apache.org/cocoon/woody/definition/1.0"
              xmlns:wi="http://apache.org/cocoon/woody/instance/1.0">
  <wd:datatype id="past-date" basetype="date">
    <!-- inline format -->
    <wd:format converter="formatting">
      <pattern locale="fr_FR">dd/MM/yyyy</pattern>
      <pattern locale="en">MM-dd-yyyy</pattern>
    </wd:format>
    <!-- validation rule -->
    <wd:validation>
      <wd:past-date>
        <wd:failmessage>Must be in the past</wd:failmessage>
      </wd:past-date>
    </wd:validation>
  </wd:datatype>

  <wd:datatype id="price" basetype="decimal">
    <!-- reference to a format, no specific validation -->
    <wd:format ref="price"/>
  </wd:datatype>

  <wd:datatype id="computertype" basetype="integer">
    <wd:selection-list>
      <wd:item value="1">
        <wi:label>PDA</wi:label>
      </wd:item>
      <wd:item value="2">
        <wi:label>Laptop</wi:label>
      </wd:item>
      <wd:item value="3">
        <wi:label>Desktop</wi:label>
      </wd:item>
      <wd:item value="4">
        <wi:label>Server</wi:label>
      </wd:item>
    </wd:selection-list>
  </wd:datatype>

  <wd:datatype id="password" basetype="string">
   <wd:validation>
     <wd:length min="6"/>
     <wd:valid-password/> <!-- check the presence of non-alpha characters -->
   </wd:validation>
  </wd:datatype>

</wd:datatypes>
```

_Open items :

- We should be able to choose between strict selection lists (enumeration) and open selection lists (an input help). This could go through an additional "open" attribute (default is false, meaning strict enumeration).
- The label in <wd:item> is in the instance namespace, as it is copied as is. Shouldn't it be the same for <wd:item> itself, and even <wd:selection-list> ?
- Which language for <wd:assert> ? XPath would allow traversal of the whole form and not be limited to the current widget's siblings._

Form definition file

The form definition file defines all widgets contained in a form.

Fields, i.e. widgets that have a value, have a datatype, which can be a reference to a datatype defined in the datatype dictionary, or define inline an anonymous datatype.

Fields can have validation rules that restrict the set of allowed values compared to that of the datatype.

Validation rules have a visibility on other fields, and also on a validation context object provided by the application. This context allows validations to perform checks related to data that is not present in the form (e.g. check that a username not already exists on a registration form). The validation context object need not (although it can) be the same as the binding context object (see binding below)

Note: it is theoretically possible for a field to have an inline datatype defining constraints and have its own constraints. Although this is not very desirable from a readability point of view, this won't cause any problem.

```
<wd:form xmlns:wd="http://apache.org/cocoon/woody/definition/1.0"
         xmlns:wi="http://apache.org/cocoon/woody/instance/1.0">

  <!-- inline anonymous datatype -->
  <wd:field id="email" required="true">
    <wd:datatype basetype="string">
      <wd:validation>
        <wd:email/>
      </wd:validation>
    </wd:datatype>
    <wi:label>Enter an <b>email</b> address:</wi:label>
  </wd:field>

  <!-- reference to an existing type -->
  <wd:field id="password" required="true">
    <wd:datatype ref="password"/>
    <wi:label>Password</wi:label>
  </wd:field>

  <!-- additional validation added to the one of the datatype -->
  <wd:field id="confirm" required="true">
    <wd:datatype ref="password"/>
    <wi:label>Confirm password</wi:label>
    <wd:validation>
      <wd:assert test="confirm = password">
        <wi:failmessage>Passwords must be equal</wi:failmessage>
      </wd:assert>
    </wd:validation>
  </wd:field>

</wd:form>
```

_Open items :

- The validation algorithm will check the datatype's rules before the field's one. How can then a field change the `<failmessage>` for a validation rule of its datatype ?
- Should `<failmessage>` be in the "wi" namespace ?
- Same remark for selection lists (although this seems less important at first)
- Can a field define its own selection list if the datatype has one ?_

Form binding

Binding the form to the

```
}}}

=  Form template =
{{{
```

Form instance

{{{
}}}