HowToEditCustomDoctypesWithBXE

Overview

Change four lines in usecase-bxeng.xmap (n.b., check the paths). This effectively allows us to specify the BXE parameters independently for each doctype. N.b., this will affect all your publications.

BXE Namespaces

```
<map:part src="../../resources/misc/bxeng/content-namespaces.xml"/>
```

to

```
<map:part src="resources/misc/bxeng/{page-envelope:document-type}-namespaces.xml"/>
```

and

BXE Config

```
<map:generate src="../../resources/misc/bxeng/inc/config.xml"/>
```

to

```
<map:generate src="resources/misc/bxeng/inc/{page-envelope:document-type}-config.xml"/>
```

and

BXE Config XSL file

```
<map:transform src="../../xslt/bxeng/config-xml.xsl">
```

to

```
<map:transform src="xslt/bxeng/{page-envelope:document-type}-config-xml.xsl">
```

and

BXE Context Menu

```
<map:parameter name="contextmenufile" value="../../resources/misc/bxeng/contextmenu.xml"/>
```

to

```
<map:parameter name="contextmenufile" value="resources/misc/bxeng/{page-envelope:document-type}-contextmenu.xml"
/>
```

n.b. you'll have to create the resources/misc/bxeng/inc and xslt/bxeng directories.

Copy ../../resources/misc/bxeng/content-namespaces.xml to resources/misc/bxeng/xhtml-namespaces.xml, ../../resources/misc/bxeng/inc/config.xml to resources/misc/bxeng/inc/xhtml-config.xml and ../../resources/misc/bxeng/contextmenu.xml to resources/misc/bxeng/xhtml-contextmenu.xml

For a new doctype called, say, "wine", copy the four xhtml- files to wine-namespaces.xml, inc/wine-config.xml, wine-contextmenu.xml and wine-config-xml.xsl respectively and modify them to suit. The namespaces file will need modifying. The config file may need changing to change the BXE toolbar buttons or callbacks. You may want to modify the context menu to exclude elements from the context menu (otherwise just remove all the xhtml specific defaults). You're unlikely to need to modify the xsl file.

How it all works

When the "Edit"->"WYSIWYG Editor (BXE)" menu item is clicked:

This menu item is defined in {pub}/config/menus/generic.xsp.

```
<menu i18n:attr="name" name="Edit" label="Search"><!-- Start of definition of "Edit" menu-->

        <block info="false">
          <item wf:event="edit" uc:usecase="kupu" uc:step="open" href="?"><i18n:text>Edit with Kupu</i18n:text><
/item>
          <!-- Next line is the BXE item-->
          <item wf:event="edit" uc:usecase="bxeng" uc:step="open" href="?"><i18n:text>Edit with BXE</i18n:text><
/item>
          <item wf:event="edit" uc:usecase="edit" uc:step="open"><xsp:attribute name="href"><xsp:expr>"?form="
+ docType</xsp:expr></xsp:attribute><i18n:text>Edit with Forms</i18n:text></item>
          <item wf:event="edit" uc:usecase="1formedit" uc:step="open" href="?"><i18n:text>Edit with one Form<
/i18n:text></item>

...

</menu>
```

The text "Edit with BXE" is a key in {pub}/../../resources/i18n/cmsui.xml and the other cmsui_nn.xml files where nn is the language code. In cmsui.xml this key is associated with the text you actually see on the menu, namely "WYSIWYG Editor (BXE)". This (English!) text is also in the other cmsui_nn.xml files by default.

The BXE item line in the code above tries to find a match inside the usecase-bxeng.xmap file for the 'open' step with a href of '?'. The usecase-bxeng.xmap file is in {pub}. You may wonder where the other usecase files for the other editors (the "One big form editor"/ 1formeditor / Source editor, the "Forms Editor" and Kupu) are. There're in {pub}/../../usecases. This makes sense as the one form editor works with any doctype and therefore doesn't need customising on a doctype or publication basis. Similarly, the only customisation likely to be needed for the multiple "Forms Editor" is in your custom doctypes' {pub}/lenya/xslt/formeditor/wine.xsl file. Kupu only supports XHTML so again is not likely to need customising. Keeping the usecases up the tree keeps them out of the way and makes sense.

The usecase-bxeng.xmap file by contrast is in {pub} precisely because we (may) need to change it.

The section in publication-sitemap.xmap begining with the comment <!-- BX Editor: RNG Schema, CSS --> is a helper section for BXE so that when it asks for wine.rng (the wine doctype's grammar file) and bxeng-wine.css (how you want the wine doctype to appear within BXE) files, Lenya will give BXE the files appropriately.

The usecase-bxeng.xmap file

The bit of the file where we start is:

```
<map:match type="usecase" pattern="bxeng">

      <map:match type="step" pattern="open">
       <!-- Check for BXENG -->
```

The usecase(="bxeng") and step (="open") match our menu item from before.

Check if BXE is installed

```
       <map:act type="resource-exists" src="../../resources/bxeng/bxeLoader.js">
```

and if it is check out our document or throw an error

```
        <map:act type="reserved-checkout">
          <map:generate type="serverpages" src="../../content/rc/{exception}.xsp">
            <map:parameter name="user" value="{user}"/>
            <map:parameter name="filename" value="{filename}"/>
            <map:parameter name="date" value="{date}"/>
            <map:parameter name="message" value="{message}"/>
          </map:generate>
          <map:transform src="../../xslt/rc/rco-exception.xsl"/>
          <map:call resource="style-cms-page"/>
        </map:act>
```

{pub}/../../resources/misc/bxeng/index.html sets up a basic BXE xhtml page that gets modified to fit with our doctype later. Again, the fact that this file is kept outside of our publication probably means we don't need to change it on a per publication basis.

{pub}/../../resources/misc/bxeng/content-namespaces.xml contains a list of the namespaces we wish to use. By default it's got xhtml, lenya, dc and dcterms defined in there. We'll need to change this file for new doctypes.

Aggregate (i.e., glue) the bxe xhtml page together with the namespaces which might seem weird (appending a bit of xml onto the end of an xhtml file) ...

```
        <map:aggregate element="bxeng">
          <map:part src="../../resources/misc/bxeng/index.xhtml"/>
          <map:part src="../../resources/misc/bxeng/content-namespaces.xml"/>
        </map:aggregate>
```

... but makes sense when you run xsl against it to insert the namespaces into meta elements (e.g., <meta name="bxeNS" content="{dc=http://purl.org/dc/elements/1.1/\}"/>) to create an xhtml document with these meta elements inside the head element as expected by BXE

```
        <map:transform src="../../xslt/bxeng/aggregate.xsl"/>
```

and add the configfile and context for our doctype to the xhtml file. (This is the important bit). N.b., The configfile and context is passed to the stylesheet. Again, the fact that the index-xhtml.xsl file is kept outside of our publication probably means we can ignore it and in fact, we can. What we're actually doing here is telling BXE to call Lenya whenever it wants a configfile or context. Note that Lenya will get called with usecase="bxeng" and step="config". I.e., call the appropriate bit of this (usecase-bxeng.xmap) file which we'll get to in a bit.

```
        <map:transform src="../../xslt/bxeng/index-xhtml.xsl">
          <map:parameter name="configfile" value="{request:requestURI}?lenya.usecase=bxeng&amp;lenya.
step=config"/>
          <map:parameter name="context" value="{request:contextPath}"/>
        </map:transform>
```

strip off the namespaces (not sure why).

```
        <map:transform src="../../xslt/util/strip_namespaces.xsl"/>
```

and output as xhtml.

```
        <map:serialize type="xhtml"/>
```

the next line is indented incorrectly (in my version) but ends the "resource-exists" (i.e., BXE is installed) action

```
        </map:act>
```

else pop up a page to tell you how to download BXE.

```
  <map:generate src="../../resources/misc/bxeng/download.xhtml"/>
  <map:call resource="style-cms-page"/>
  <map:serialize type="xhtml"/>
</map:match>
```

We've not had to modify anything so far which is why other documentation on BXE says "start with the config step".

An image upload dialog box step ...

```
<map:match pattern="image-upload-show" type="step">
  <map:call resource="cms-screen">
     <map:parameter name="serverpage" value="info/assets.xsp"/>
     <map:parameter name="stylesheet" value="bxeng/image.xsl"/>
  </map:call>
</map:match>
```

An asset upload dialog box step ...

```
<map:match pattern="asset-upload-show" type="step">
  <map:call resource="cms-screen">
     <map:parameter name="serverpage" value="info/assets.xsp"/>
     <map:parameter name="stylesheet" value="bxeng/asset.xsl"/>
  </map:call>
</map:match>
```

Trigger an asset upload... (calls the asset upload dialog box step above)

```
    <map:match type="step" pattern="asset-upload">
      <map:act type="upload">
        <map:redirect-to uri="{request:requestURI}?lenya.usecase=bxeng&amp;lenya.step=asset-upload-show"/>
      </map:act>
        <map:generate src="../../content/info/assets.xsp" type="serverpages"/>
         <map:transform src="../../xslt/bxeng/asset.xsl">
           <map:parameter name="use-request-parameters" value="true"/>
           <map:parameter name="error" value="true"/>
         </map:transform>
         <map:call resource="style-cms-page"/>
    </map:match>
```

Trigger an image upload ... (calls the image upload dialog box step above)

```
    <map:match type="step" pattern="image-upload">
      <map:act type="upload">
        <map:redirect-to uri="{request:requestURI}?lenya.usecase=bxeng&amp;lenya.step=image-upload-show"/>
      </map:act>
        <map:generate src="../../content/info/assets.xsp" type="serverpages"/>
        <map:transform src="../../xslt/bxeng/image.xsl">
          <map:parameter name="use-request-parameters" value="true"/>
          <map:parameter name="error" value="true"/>
        </map:transform>
        <map:call resource="style-cms-page"/>
         </map:match>
```

Sort out links ...

```
    <map:match pattern="link-show" type="step">
          <!-- just a dummy xsp since we call the info area directly -->
         <map:generate type="serverpages" src="../../content/info/assets.xsp"/>
        <map:transform src="../../xslt/bxeng/link.xsl" label="content">
          <map:parameter name="infoarea" value="true"/>
          <map:parameter name="contextprefix" value="{request:contextPath}"/>
          <map:parameter name="publicationid" value="{page-envelope:publication-id}"/>
          <map:parameter name="area" value="authoring"/>
          <map:parameter name="tab" value="en"/>
          <map:parameter name="chosenlanguage" value="{page-envelope:document-language}"/>
          <map:parameter name="documentid" value="{page-envelope:document-id}"/>
          <map:parameter name="documenturl" value="/{page-envelope:document-url}"/>
          <map:parameter name="documentextension" value="{page-envelope:document-extension}"/>
          <map:parameter name="defaultlanguage" value="{page-envelope:default-language}"/>
          <map:parameter name="languages" value="{page-envelope:publication-languages-csv}"/>
        </map:transform>
      <map:call resource="style-cms-page"/>
    </map:match>
```

BXE will now call this step below as set up earlier.

```
    <map:match pattern="**/*.html">
      <!-- configuration -->
      <map:match type="step" pattern="config">
```

Grab the config.xml file. This is a standard BXE config.xml file which tells BXE what input files to use (the xml document file we want to edit, the xhtml file we created earlier, the doctype's rng file and the xsl file to turn the xml file into the output file), what output we want, which bxe scripts to load and what buttons we want in our interface (e.g., lenya asset upload buttons are included here).

```
        <map:generate src="../../resources/misc/bxeng/inc/config.xml"/>
```

An xsl file is run against the config.xml file with parameters to transform it into one suitable for our doctype.

```
        <map:transform src="../../xslt/bxeng/config-xml.xsl">
```

Tell BXE to call Lenya usecase="bxeng", step="xml" whenever it wants the xml input file

```
          <map:parameter name="BX_xmlfile" value="{request:requestURI}?lenya.usecase=bxeng&amp;lenya.step=xml"
/>
```

Insert our default language.

```
          <map:parameter name="defaultlanguage" value="{page-envelope:default-language}"/>

  <!--      Instead of an xsl we use the xhtml file to provide the basic layout
          <map:parameter name="BX_xslfile" value="{2}.xsl"/>
  -->
```

Not sure about this one, I think it gets matched in publication-sitemap.xmap(?).

```
          <map:parameter name="BX_xhtmlfile" value="{../2}.bxe.html"/>
```

Pull in the correct rng based on doctype (using the rng bit we mentioned earlier in publication-sitemap.xmap).

```
            <map:parameter name="BX_validationfile" value="{request:contextPath}/{page-envelope:publication-id}/
{page-envelope:area}/{page-envelope:document-type}.rng"/>
```

Pull in the correct css based on doctype (using the css bit we mentioned earlier in publication-sitemap.xmap).

```
            <map:parameter name="css" value="{request:contextPath}/{page-envelope:publication-id}/{page-
envelope:area}/css/{page-envelope:document-type}-bxeng.css"/>
  <!--       The document is checked in when we exit from bx (in case of save&exit and in case of exit), so we
use the usecase
             for the checkin while we redirect to the document
  -->
```

Tell BXE to call Lenya with usecase="checkin", step="checkin" when it exits.

```
            <map:parameter name="BX_exitdestination" value="{request:requestURI}?lenya.usecase=checkin&amp;
lenya.step=checkin&amp;backup=true"/>
```

Set up a context menu (might want to change this)

```
            <map:parameter name="contextmenufile" value="../../resources/misc/bxeng/contextmenu.xml"/>
          </map:transform>
          <map:transform type="cinclude"/>
```

Outout our modified config file.

```
          <map:serialize type="xml"/>
        </map:match>
      </map:match>
```

This is the usecase="bxeng", step="xml" that we told BXE to call to get its xml input file

```
      <!-- /GET and PUT -->
      <map:match type="step" pattern="xml">
        <map:select type="request-method">

          <map:when test="PUT">
            <!-- before we save, we must be sure that the document is well checked out
            -->
```

Are we really checked out? Throw an error if not.

```
            <map:act type="reserved-checkout-test">
              <map:generate type="serverpages" src="../../content/rc/{exception}.xsp">
                <map:parameter name="user" value="{user}"/>
                <map:parameter name="filename" value="{filename}"/>
                <map:parameter name="date" value="{date}"/>
              </map:generate>
              <map:transform src="../../xslt/rc/rco-exception.xsl"/>
              <map:call resource="style-cms-page"/>
            </map:act>

        <map:call function="editDocument">
          <map:parameter name="sourceUri" value="cocoon:/request2document"/>
          <map:parameter name="noCheckin" value="true"/>
        </map:call>
        </map:when>

        <map:otherwise> <!-- GET -->
```

Fetch our document.

```
          <map:generate src="content/authoring/{page-envelope:document-path}"/>
```

Change xhtml:object element paths.

```
          <map:transform src="../../xslt/bxeng/change-object-path.xsl">
            <map:parameter name="documentid" value="{page-envelope:document-id}"/>
          </map:transform>
```

Serialize as XML and hand it to BXE.

```
          <map:serialize type="xml"/>
        </map:otherwise>

      </map:select>
    </map:match>
    <!-- /GET and PUT -->

    </map:match> <!-- uri pattern -->
```

Questions

1. Why do the namespaces need to be stripped?