Part 5: Custom Navigation in Lenya

Table of contents

OK, so you have the basics: you know how to install Lenya, you understand it's approach to content management, you understand the pipeline bit, and you know how to edit existing documents. Through playing around, you probably now know how to make new pages, and create the content hierarchy for your site. Now the question becomes, how do we customize the navigation to be marked up the way you want.

1. Customizing Lenya navigation items

In order to understand customizing navigation items in Lenya, you'll need to understand where that's possible. In the default publication's root directory, you'll see a directory called 'lenya'. It's here where you can add folders and files that override the global defaults set aside for all of Lenya.

One of the directories is 'navigation'. In there, you'll most likely see a file called tabs.xsl. This is a customized version of the tabs just for the default publication. But you can do others. Here's the list of the file names you can add in the 'navigation' folder that will override the global files:

breadcrumb.xsl	The breadcrumb trails at the top of the page (e.g. Home > Products > Furniture)
menu.xsl	The menu at the left of the pages (in the default publication, for example)
tabs.xsl	The tabs at the top of the page (in the default publication, they are the highest levels of nav on the site)
search.xsl	The search box on the site

There are others, but these will do for now. You can check out what the originals are in /usr/local/tomcat/webapps/lenya/lenya/xslt/navigation/. It requires some understanding XSLT, which is again beyond the scope of this article. W3Schools (http://www.w3schools.com/xsl/default.asp) has a nice place to learn XSLT quickly.

We're going to take a look at the sitetree structure to find out what you're capable of knowing about an item in the navigation of the site, then do a quick example of how to create your own menu.

2. Understanding the site tree

The site tree is the place where all the "nodes", or pages in your publication are stored, preserving the hierarchy. The XML file can be found by going to the content/authoring/ directory in the default publication and viewing the sitetree.xml file. Let's take a look at a chunk:

```
<site>
    <node id="index">
        <label xml:lang="en">Home</label>
        <label xml:lang="de">Home</label>
    </node>

    <node id="tutorial">
        <label xml:lang="en">Tutorial</label>
        <label xml:lang="de">Tutorial</label>

            <node id="new_doctype">
                <label xml:lang="en">Create new doctype</label>
            </node>
    </node>
</site>
```

The above tells us that each page is characterized by a <node> tag. Each <node> tag has an "id" attribute, which is the name of the document. So, if this publication's address were http://www.someplace.com/, then the ID for the second node would make

the address to page http://www.someplace.com/tutorial.html.

Each node has an inner element called <label>. The label tag's contents are the navigation title for the page. It's the link to the page that you see in menu of the site. The xml:lang attribute signifies the language for this label, and as you can see, this site has multiple supported languages: "en" for English, and "de" for German.

Nodes can be inside other nodes. In the example above, the "new_doctype" node is inside the "tutorial" node. This means that the "tutorial" page is a parent of the "new_doctype" page, so they fall underneath each other like so:

```
1. Home
2. Tutorial
    a. Create new doctype
```

Order is also important - the order of the nodes in the sitetree file is the order in which the pages are displayed in the hierarchy of the site.

3. The nav: namespace

An XML namespace was created for users to grab these nodes with the appropriate information. So, for example, you can reference a node in your XSL using nav:node. Once you reference the node, you can reference it's label too: nav:label. There's some other items you can reference when you are pointing at a specific node:

@href	The location of the page relative to where you are currently in the site
@current	A true or false value is possible to determine whether or not the page you are on is the node you are referencing
@visibleinnav	Another true/false value given if the page had been labeled as visible in the navigation

Hiding navigation items can be helpful for situations like form submission/confirmation pages. You don't want the user to go to that page right away from within the site's menu, so you hide it, letting the programming of the form's input redirect the user to that page instead. Other examples abound.

4. Using your new-found knowledge

Let's try putting together a simple menu.xsl file that creates our own simple markup. The contents are below:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:nav="http://apache.org/cocoon/lenya/navigation/1.0"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="nav">

<xsl:template match="nav:site">

    <ul>

    <!-- loop through all top-level items and return as list items -->
    <xsl:for-each select="/*/nav:node[@visibleinnav = 'true']">
        <xsl:when test="@current = 'true'">
            <li class="current"><a href="{@href}"><xsl:apply-templates
select="nav:label"/></a></li>
        </xsl:when>
        <xsl:otherwise>
            <li><a href="{@href}"><xsl:apply-templates select="nav:label"/></a></li>
```

```
            </xsl:otherwise>
        </xsl:for-each>

        </ul>
    </xsl:template>

    <xsl:template match="nav:label">
        <xsl:value-of select="."/>
    </xsl:template>

    </xsl:stylesheet>
```

OK, so let's dissect this one. You always start with your XML prologue and stylesheet tags (notice how the root stylesheet tag references the nav namespace?). Then, one template is created to match the <site> tag - that's the root tag that wraps around all the <node> tags in our sitetree.xml file.

We open up the unordered list, and then we'll just loop through the top-level navigation items on the site. The '/*/nav:node' part of the for-each loop steps down into the site tag that you just matched (/*), and then through the nodes underneath the site tag (/nav:node). Note that this is only the top level nodes, not all the parents and children! Read up on XSLT (http://www.w3schools.com/xsl/default.asp) and XPath (http://www.w3schools.com/xpath/default.asp) to better understand how this notation works.

For each top-level item, we test to see that node happens to be the page we are looking at right now (@current = 'true'). If it is, then we open up our list tag and assign it the class of "current". Your CSS can then be created to give some special styling to that list item.

Once the list item is created, we need the link. The address for the link tag is gathered from our handy @href. The text for the link will be the label of the node. So, we call out another template shown at the bottom of our example menu.xsl file where it basically just grabs the value of the contents inside the <label> tags in the sitetree.xml file. We close the the link and the list items.

Of course, if this isn't the page we are on, then we do the same thing but don't add in our special class. Simple, no?

5. Give it a try

You now pretty much have the basics for putting together your own custom navigation. There are those that feel our methods for generating the menus are incorrect and that we should grab the nodes that we need in one fell swoop instead of looping through each one, but this was the only method we could find where wewould achieve the coding standards we wanted. Go experiment and have fun!