Extra functionality for finding aids on the Web

Written 2009 by Joyce Chapman
NCSU Libraries Fellow
Metadata & Cataloging/Digital Library Initiatives
Contact joyce_chapman@ncsu.edu

After several people on the EAD-listserv wrote and asked if I could explain how to produce collapsible section, automated hyperlinks in the <arrangement> section, or alternating colors in the Container List, I decided to write up step-by-step instructions for these three functionalities and share them with a broader audience. Please note that there are various ways to implement all three of these functionalities in a finding aid Web display: I have only described the way that they were implemented at the Southern Historical Collection at UNC-Chapel Hill in 2009.

Who are these instructions for? You will need very basic familiarity with XSLT and XPATH to implement these functionalities. In your stylesheet you must be able to locate the template that affects a particular EAD element, and you must be able to modify the XPATH that I give you in the event that your templates are set up different from the examples here. Feel free to email me at joyce_chapman@ncsu.edu if you have trouble.

Zip file contents. This PDF file should have been downloaded together with an example XML file and HTML file. You can see examples of all this code in action in the HTML file, and view the HTML's original XML in the XML file. If you do not have the helper files, please download the original zip file from the EAD Helper Tools page: http://www.archivists.org/saagroups/ead/tools.html

The JavaScript for collapsible sections was written by Adrienne MacKay and Stephanie Adamson of the Carolina Digital Library and Archives in 2007/2008.

COLLAPSIBLE SECTIONS

Requires:

* Modification of css
* Modification of XSL in four places

The hardest one first! The method detailed below for collapsing sections uses JavaScript (see example, eadexample.html). This method allows for "collapse all" and "expand all" buttons as well as the collapsing and expanding of individual sections. Sections collapse when an image is clicked. When clicked, the image will also change into another image (for example, a side-facing arrow will turn into a down-facing arrow). There are a number of adjustments that must be made to your stylesheet in order to implement these collapsible sections, so have patience!

Step 1.

In order to create the "collapse all" and "expand all" buttons, cut and paste the following two links in your XSLT stylesheet. You can put them wherever you want, based on where you want them to appear in the HTML (though if you use frames, I don't think it would work to put these in a frame other than the one with the sections that collapse). You can also choose to leave this chunk of code out entirely if you do not want "expand all" and "collapse all" sections.

Things you need to modify in the code:

1. Text in black. This is what appears as the text of your link.
2. The image paths in red, ex. path/to/image/representing/expanded/sections.png. You will change these to link to the images you have chosen.

```
<a href="#" onclick="expandAll();">[expand all sections
        <img src="path/to/image/representing/expanded/sections.png"
        alt="expand/collapse"></img>]
</a>
<a href="#" onclick="collapseAll();">[collapse all sections
        <img src="path/to/image/representing/collapsed/sections.png"
         alt="collapse arrow"></img>]
</a>
```

Step two

Cut and paste the following chunk of JavaScript into your XSLT stylesheet as the *last thing* before the closing body tag (`</body>`) of your HTML.

Things you need to modify in the code:

1. The image paths in red, ex. path/to/image/representing/expanded/sections.png. You will change these to link to the location to link to the images you have chosen.

```
<script type="text/javascript">
	//define an array identifying all expandible/collapsible divs by ID
	//this is limited to div elements.
	var divs=document.getElementsByTagName('div');
	var ids=new Array();
	var n=0;
	for (var i=0;i<divs.length;i++){
	//conditional statement limits divs in array to those with class 'showhide'
	//change 'showhide' to whatever class identifies expandable/collapsible divs.
	if (divs[i].className=='showhide') {
	ids[n]=divs[i].id;
	n++;
	}
	}

	function expandAll(){
	//loop through the array and expand each element by ID
	for (var i=0;i<ids.length;i++){
	var ex_obj = document.getElementById(ids[i]).style;
	ex_obj.display = ex_obj.display == "none" ? "block" : "block";
	//swap corresponding arrow image to reflect change
	var arrow = document.getElementById("x"+(ids[i]));
	arrow.src="path/to/image/representing/collapsed/sections.png";
	}
	}

	function collapseAll(){
	//loop through the array and collapse each element by ID
	for (var i=0;i<ids.length;i++){
	var col_obj = document.getElementById(ids[i]).style;
	col_obj.display = col_obj.display == "block" ? "none" : "none";
	//swap corresponding arrow image to reflect change
	var arrow = document.getElementById("x"+(ids[i]));
	arrow.src="path/to/image/representing/expanded/sections.png";
	}
	}

	// show/hide script for individual div toggling
	  //by Adrienne MacKay, adapted 2008 by Stephanie Adamson to include arrow
	  image swapping
	function changeDisplay(obj_name) {
	var my_obj = document.getElementById(obj_name);
	var arrow=document.getElementById("x"+obj_name);
	if (my_obj.style.display=="none") {
	my_obj.style.display="block";
```

```
        arrow.src="path/to/image/representing/expanded/sections.png";
        }
        else { my_obj.style.display="none";
        arrow.src="path/to/image/representing/collapsed/sections.png";
        }
        }
</script>
```

Step three

The tedious part. For every section of the finding aid that you wish to be expandable and collapsible (ex., every series in the `<dsc>`) you need to add a couple different things to the HTML output. Just for clarity before diving into XSLT, let's take a look at how we want our HTML output to appear (the bright pink text, red text, and black text will be replaced with code generated by you):

Line	HTML
1	`<h3><a href="javascript:changeDisplay('ID');">`
2	`<img id="xID" src="path/to/image/representing/expanded/sections.png"`
3	`alt="expand/collapse"/></a>`
4	`TITLE OF SECTION</h3>`
5	`<div id="ID" class="showhide" style="display: block;">`
6	`<p>I'm in a div that collapses and expands!!</p>`
7	`</div>`

Each collapsible/expandable section must be enclosed in a `<div>` with a unique ID (line 5-7). This same unique ID will be referenced by the JavaScript in the `<a>` tag (line 1) and the id attribute in the `<img>` tag (line 2). Each collapsible div additionally needs the following two attributes and values: class="showhide" style="display: block;" (line 5). The `<img>` should be enclosed in the link tag `<a>` and the link tag must include the following attribute and value href="javascript:changeDisplay('ID');" (line 2).

How can you mess this part up? You could output the `<h3>` and the `<img>` inside the collapsible `<div>` instead of above it. If you do that, when a user collapses the section, not only with the content disappear, the section header and the mechanism that allows section to expand again will also disappear.

Now to modifying the XSLT. First, we'll modify the section header and add the image link. Locate the places in your XSLT that are already outputting the section headers (you probably have this in a number of different templates, maybe one place in a template for Subject Headings, another in a template for series within the Container List, etc.) Swap out the simple code outputting your section header with the following chunk of code. For the example, let's pretend like your current code is this:

```
<h3>
   <xsl:value-of select="head"/>
</h3>
```

The new code you need to add into the old code looks like this:

```
<h3>
<xsl:element name="a">
            <xsl:attribute name="href">
               <xsl:text>javascript:changeDisplay('</xsl:text>
               <xsl:value-of select="generate-id(current())"/>
               <xsl:text>');</xsl:text>
            </xsl:attribute>
         <img id="x{generate-id(current())}"
               src="path/to/image/representing/expanded/sections.png"
         alt="expand/collapse"/>
         </xsl:element>
         <xsl:value-of select="head"/>
</h3>
```

Things you need to modify in the code:

1. Text in black. This is what appears as the text of your link.
2. The image paths in red, ex. path/to/image/representing/expanded/sections.png. You will change these to link to the location to link to the images you have chosen.
3. The `<h3></h3>`. This should be whatever level header your institution uses.

To be most efficient, you could create a called template and call it in each place that you output a section header. But if you just want to stick this chunk in over and over again, that's ok too.

How can you mess this part up? You'll notice that we're using `"generate-id(current())"`[1] to get that ID that repeats in several different places. Just make sure that you are generating the ID for the *same node*. In the code above, we are generating two of the three necessary instances of that ID, and we are generating it on the context node.[1] If the current node for the above code is different from the current node in the next part, your sections will not collapse.

Next, find the places that output the content of the sections you want to collapse. This can be really hard. You may already have the entire section (header and content) enclosed in a `<div>`; however, we need a `<div>` that encloses *only* the content to be collapsed and not the header. Enclose the part of your code that outputs the section contents (probably a simple "<xsl:apply-templates/>" or perhaps specific template applications, "<xsl:apply-templates select="controlaccess"/>) and you need to encase that in a `<div>`:

---

[1] See the last page of this document for an explanation of "generate-id" and "context node"

```
<div id="{generate-id(path/to/ID/node)}" class="showhide" style="display: block;">
        <xsl:apply-templates/>
</div>
```

IMPORTANT: you must replace the XPATH in pink above with the XPATH to the same element for which you generated an ID in the first chunk of code. If you are now working within a different template, this XPATH might be different.

TIP: if you get seriously hung up trying to generate the same IDs in all three of these places and it is simply not generating the same ID, you've got the wrong path in one of the places. Try using the `name()` function to output the name of the node into your HTML so you can take a look at it:

```
<div id="{name(path/to/ID/node)}" class="showhide" style="display: block;">
        <xsl:apply-templates/>
</div>
```

Then run your XML through the stylesheet and take a look at the resultant HTML. Whatever value appears in the div ID is the node for which your XPATH is actually generating an ID. You may find that you thought you were outputting an ID for something other than what you are. Seeing the results will point in the right direction of how to fix it.

Now everything should work!

HYPERLINKED SERIES QUICK LINKS

Requires:

* Modification of css
* Modification of XSL in three places

One very easy way to create a hyperlinked table of contents of your collections series and subseries (see example, eadexample.html) is to use XSLT to automatically produce the hyperlinked section in the HTML. If you already had an `<arrangement>` section encoded, you can tell your XSLT to suppress the `<list>` inside `<arrangement>` while continuing to output the narrative information encoded in `<p>`, and output your new and improved hyperlinked list instead of the old list. Besides allowing for easy hyperlinking of sections, this method reduces problems caused by inconsistencies in hand-coded `<arrangement>` sections (such as incorrectly numbered series or inconsistencies in number of levels of subseries that were encoded). Using the method described below, you may choose whether or not to display the series dates alongside the series titles in hyperlinks, and you may choose how many levels of subseries you want to display in the hyperlinked table of contents. You may also choose when to trigger the appearances of the hyperlinked Series Quick Links: for example, the code below only outputs the hyperlinked section if there is more than one series present in the `<dsc>`.

To create the hyperlinked Series Quick Links, you need code in three places in your XSLT. First, suppress the encoded `<arrangement>`; second, output the new hyperlinked section; third, create anchor links at series/subseries heads to which the hyperlinks can link.

Step one

Empty out the current template for `<list>` inside `<arrangement>` (note that if you encode the list of series in some other tag, you should suppress that instead of `<list>`). Don't erase the template, just cause the encoded list within to do nothing, like this:

```
<xsl:template match="ead/archdesc/scopecontent/arrangement/list">
</xsl:template>
```

If there is no template that looks like the above, you may have a general template that applies to all `<list>` tags. Just add the template above to your stylesheet: it will override the application of the more general template. Make sure to check that the XPATH is aligned with your institution's EAD.

Step two

Add the following code within the template that applies to your `<dsc>` to produce a new Series Quick Link section. Please note that you may need to modify the XPATH selections used below based on your institution's use of the component attribute "level"

(for example, if you use lots of record groups or subgroups that should appear in the list, those are unaccounted for here).

```
<!-- Only create this section if there is more than one series -->
<xsl:if test="count(c01[@level='series']) > 1">
<h3>Series Quick Links</h3>
<ul>
<!-- for every c01 that is a subgrp or series, or c02-c04 that is a subseries, create a hyperlink. Add or delete here in order to make more or fewer levels appear in your Series Quick Links -->
<xsl:for-each
select="c01[@level='series']/did/unittitle | c01/c02[@level='subseries']/did/unittitle | c01//c03[@level='subseries']/did/unittitle | c01//c04[@level='subseries']/did/unittitle">
<li>
   <xsl:element name="a">
         <xsl:attribute name="class">x<xsl:value-of select="count
         (ancestor::*)"/></xsl:attribute>
         <xsl:attribute name="href">#<xsl:value-of select="generate-
         id()"/></xsl:attribute>
              <xsl:value-of select="."/>
   </xsl:element>
</li>
</xsl:for-each>
 </ul>
</xsl:if>
```

(If you're interested, see the last page of this document to understand why I'm counting ancestor nodes in the above code).

The above code will output HTML that looks something like this:

```
<ul>
    <li><a class="x5" href="#d1e143">[Series 1. xxx]</a>
    </li>
    <li><a class="x6" href="#d1e158">[Series 1.1. xxx]</a>
    </li>
       ...etc. for every series/subseries
 </ul>
```

The reason you are outputting an "x" in front of the number is because it is invalid to begin an attribute value with a number in HTML. The "x" has no other meaning.

In order for your newly created links to have something to link to, you'll need to also create an anchor at the top of each series in the Container List, with an ID that matches the "href" generated above. In this example, we generated a hyperlink to an anchor ID for each series and subseries component's `did/unittitle`. Find the template in your

stylesheet that outputs the `did/unittitle` of series in the container list. You can either make the `did/unittitle` itself an anchor, or you can create an empty anchor right in front of it. Modify the XPATH-selector in the example below based on where this code is going to be in your stylesheet in relation to the element for which you are generating an ID. In this example, I am working from within a template that applies to componant level nodes, so the XPATH specifies that I want to generate an ID for series and subseries-level `did/unittitle` "generate-id(did/unittitle)". In the code that generates the linked sections above, we were working from within a for-each statement that applied to series and subseries-level `did/unittitle`, so we generated an ID for the *current node,* "generate-id()". In both cases, we generated IDs for the same node, just from different starting places. Enough talk, here's the code:

```
<xsl:element name="a">
      <xsl:attribute name="name">
         <xsl:value-of select="generate-id(did/unittitle)"/>
      </xsl:attribute>
 </xsl:element>
```

Lastly, add the following to the CSS to your CSS stylesheet in order to indent each level of subseries hyperlinks. Adjust the margins as you see fit:

```
.x6 {
   margin-left: 15px;
}
.x7 {
   margin-left: 30px;
}
```
...etc for as many levels of indentation as you have created

ALTERNATING COLORS IN CONTAINER LIST

Requires:

* Modification of css
* Modification of XSL in one place

This is extremely easy to achieve with CSS, as long as your XSLT is outputting the contents of each component level in its own table row. If you are outputting the contents of a single component in multiple table rows (for example, a component's containers and `<did>` in one row and its `<scopecontent>` in the next row) this will not work.

To cause HTML table rows to alternate color, the rows, or `<tr>` elements, need alternating CSS classes. The tricky part is how to get your XSLT to output alternating classes on table rows. Within the template for container-level components, we will declare a variable and use the `mod` arithmetic operator to calculate the component's position (odd or even) in relation to all its sibling components. The `mod` arithmetic operator finds the remainder after integer division: it divides an element's position within a series of sibling elements by two, and outputs the remainder (which will always be 1 or 0). If the remainder is 1, this will identify all odd-numbered nodes, 0 will identify all even-numbered nodes. We are able to add HTML class attributes to table rows based on whether the position of the element in relation to sibling elements is "even" or "odd."

Exactly how you incorporate this into your XSLT stylesheet will depend on how your templates are set up. In the example below, we are working within a template that applies to the container-level component. If you do not have your templates set up similarly, modify the code below.

All we need to do is modify the part of the old XLST code that outputs the table rows, which should look something like:

```
<tr>
```

Locate this part of the container-level component template. Add this:

```
<tr class="{$class}">
```

Next, you need to add the variable that you used in the above code (the dollar sign means that you are dealing with a variable named "class"). The chunk of code below needs to go in a very specific place. Either you need to insert it as the first thing inside the template where the above-mentioned table rows are being output (example: right after <xsl:template match="*[@level='file']">), or if you are outputting those table rows within a <xsl:for-each> statement, add this code as the very first thing inside the statement (example: right after <xsl:for-each select="*[@level='file']">).

```
<xsl:variable name="class">
    <xsl:choose>
```

```
            <xsl:when test="position() mod 2 = 0">mod0</xsl:when>
            <xsl:otherwise>mod1</xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
```

Then insert the following CSS into your CSS stylesheet. You'll want to change the colors☺

```
.mod1 {
   background-color:white
}
.mod0 {
   background-color:pink
}
```

What's generate-id()?
You are using the `generate-id()` function to generate a unique ID for a given node within your XML document. This function will supply the same ID for a given node no matter where or how many times you use it in your XSLT. Say the ID for finding aid XYZ's `<dsc>` node is d43098. No matter what template I'm working inside in my XSLT for finding aid XYZ, if I generate an ID on the element `<dsc>`, it will come out to be d43098.

What's count(ancestor::*)?
This function counts the number of XML elements inside of which the current node is nested. We are outputting the number as an attribute on each series hyperlink, so we can use CSS to indent each level of subseries in the list further than the last (for a demonstration, view the file eadexample.html in your browser). For any given finding aid, all series at the same level will have the same number of ancestors. So series 1, 2 and 3 may have six ancestor nodes, and subseries 1.1, 1.2, and then 3.4 would all have seven ancestor nodes.

What's the context node?
The context node is the source document node currently being processed by the XSLT processor as specified by a template. So if you are writing code inside a template that matches `ead/archdesc/descgrp/userestrict`, the context node is `userestrict.` You can use the shorthand `"."` to represent the context node in any template. If you were in a template that matched `ead/archdesc/bioghist` and used the shorthand `"."` you would be referring to the context node `bioghist`.