Introduction to XSLT: the XML Transformation Language

Alan Houser
Principal Consultant and Trainer

Tel: 412-450-0532
arh@groupwellesley.com
www.groupwellesley.com

tekom jahres tagung 2015
STUTTGART, 10.–12. NOV.

tcworld conference 2015
STUTTGART, NOVEMBER 10–12

Your opinion is important to us! Please tell us what you thought of the lecture. We look forward to your feedback via smartphone or tablet under

http://IN50.honestly.de

or scan the QR code

The feedback tool will be available even after the conference!

What you will learn

- Basic XSLT stylesheet structure and instructions
- How to convert XML to XHTML for publishing
- How to publish customized documents by selecting particular chunks of XML content
- How to create hyperlinks for cross-references and navigation
- How to simplify your XSLT stylesheets by using CSS

Why XSLT?

- XSLT is a W3C-standard programming language for manipulating XML documents.
- XSLT, along with its companion language XPath, allows you to transform your XML content to a form expected by your publishing engine; whether print, PDF, HTML, wireless device, or any other current or future publishing format.
- XSLT provides the capability to select, sort, or filter XML content, allowing customized publishing based on XML metadata.

History and terminology

- **XSL** – Extensible stylesheet language: original W3C working draft recommendation
- **XSLT** – Extensible stylesheet language transformations
- **XPath** – XML path language
- **XSL-FO** – Extensible stylesheet language formatting objects

About this Workshop

During this workshop, we will learn enough XSLT to transform XML to HTML for publishing. Some knowledge of HTML will be helpful.

These tasks and techniques apply to transforming any XML documents (e.g. DITA, Docbook) to any other target vocabulary (e.g., XSL-FO).

About this Workshop

We will use:

- Notepad, or your favorite text editor
- MacOS: Safari Browser
- Windows: Internet Explorer

Our Workflow

- Extract sample files to any convenient location on your system.
- Each XML document references a stylesheet:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="navigation_working.xsl"?>
```

- When you load an XML document into Internet Explorer, IE applies the specified XSLT stylesheet to the XML document and displays the result.

Anatomy of an XSLT Stylesheet

An XSLT stylesheet is a well-formed XML document.

- All elements must be closed.
- All attribute values must be quoted.

Required root element <xsl:stylesheet>.

XSLT instruction elements prefixed by “xsl:”

Literal result elements (e.g., HTML tags) have no namespace prefix.

About our XML Data

- We will work with a set of (DITA-compliant) tasks.
- Open task_no_stylesheet.xml and examine the XML structure.

Minimal XSLT Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" />

<!- match all elements, write their text content -->
<xsl:template match="*" >
  <xsl:apply-templates />
</xsl:template>

</xsl:stylesheet>
```

Exercise: Minimal XSLT Stylesheet

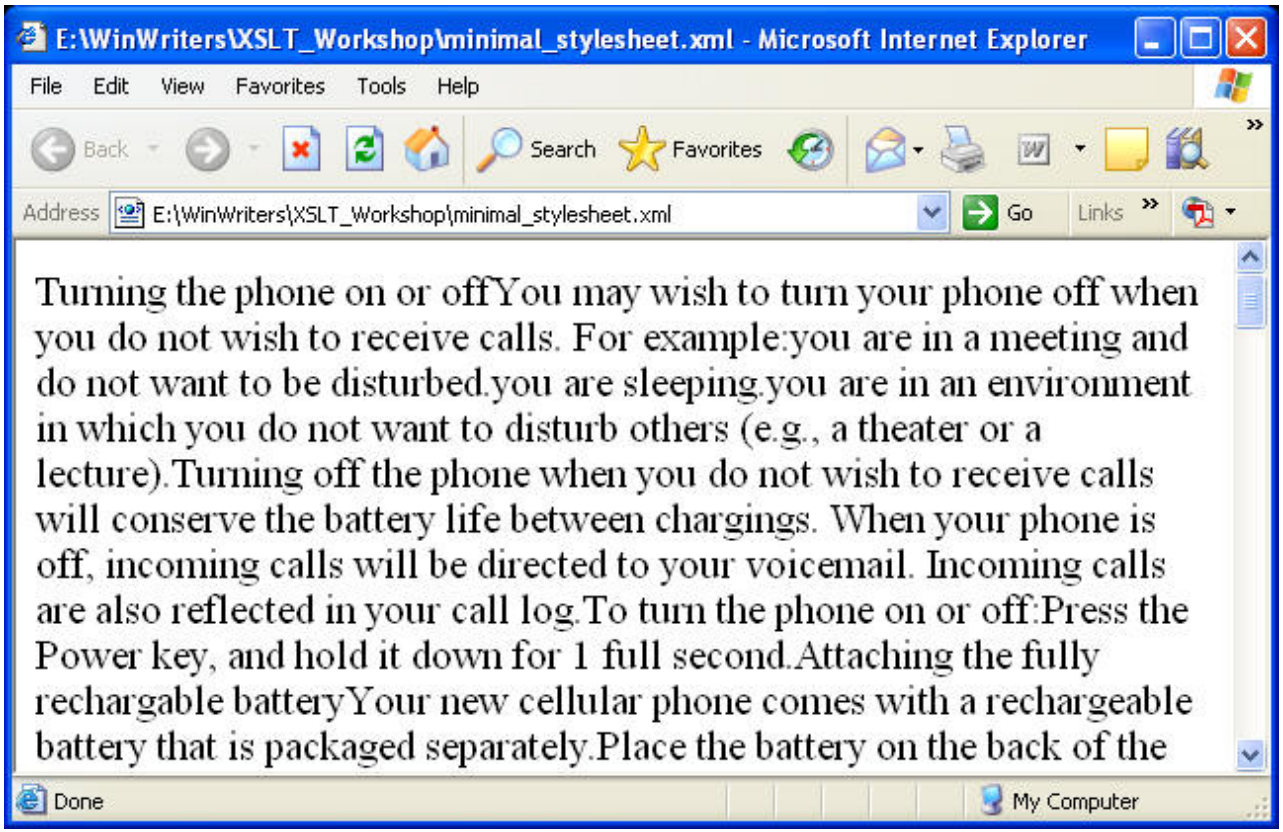
Open minimal_stylesheet.xml. This applies the previous simple XSLT stylesheet to our XML data, and displays the result in the browser.

Note that only the content of the original XML document is passed through by the stylesheet. All original element boundaries (tags) are dropped.

We will add HTML tags in the next exercise.

Exercise: Minimal XSLT Stylesheet

Result of opening minimal_stylesheet.xml in Internet Explorer.



Exercise: Minimal XSLT Stylesheet (2)

Use “View Source” to see the result of the XSLT transformation. This is what the browser is rendering.

Task: Converting XML to XHTML

- Approach: Map XML elements to HTML elements.
- Use <xsl:template match="*element name*"> for each XML element to be converted.
- Use <xsl:apply-templates /> to continue processing child elements.
- Example: Convert <body> to <p>:

```
<xsl:template match="body">
    <p></xsl:apply-templates /></p>
</xsl:template>
```

Task: Converting XML to XHTML

Use these mappings for the provided sample files. Note that names are case-sensitive.

XML	HTML
title	h1
cmd	p
p	p
info	p

XML	HTML
steps	ol
step	li
li	li
ul	ul
uicontrol	em
tasklist	html/body

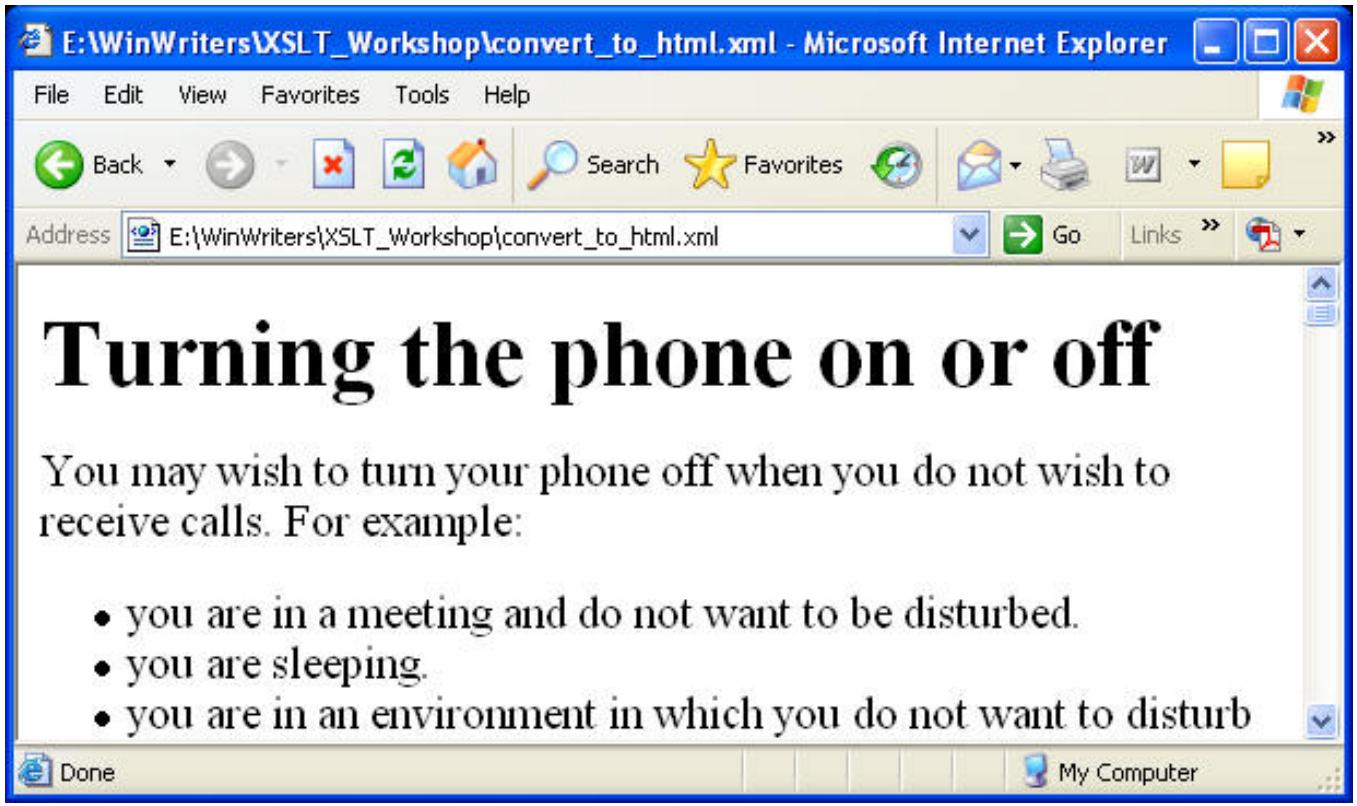
Exercise: Converting XML to XHTML

Modify convert_to_html.xsl to convert XML elements to HTML. Use <xsl:template> and <xsl:apply-templates/> for each element to be converted.

As you work, open convert_to_html.xml in a browser to view your progress.

Exercise: Converting XML to XHTML

Result of opening convert_to_html.xml in Internet Explorer.



Task: Combining XSLT and CSS

- I know XSLT, but my designer knows CSS. How can I design an XSLT stylesheet that will leverage both of our skill sets?
- Our corporate Web design guidelines are strict, change regularly, and are specified as a cascading stylesheet. How can I design my XSLT stylesheet so that my HTML will automatically pick up changes in our Web site CSS file?

Task: Combining XSLT and CSS

To separate “transformation” from “formatting”, specify a CSS stylesheet link in the result HTML document.

```
<xsl:template  match=“tasklist">
   <html>
   <head>
   <link  rel="StyleSheet"  href=“task.css“  type="text/css"
   media="screen"/>
   </head>
   <body>
        <xsl:apply-templates  />
   </body>
</xsl:template>
```

A Little Bit of CSS

```
body {display:block; font-family:'Helvetica, sans-serif'; }
```

CSS (particularly CSS3) provides some of the capabilities of XSLT. Examples:

- Select element based on its context.
- Show/hide elements.

But XSLT supports much richer transformations, like changing one XML vocabulary to another or sorting XML content.

Exercise: Combining XSLT and CSS

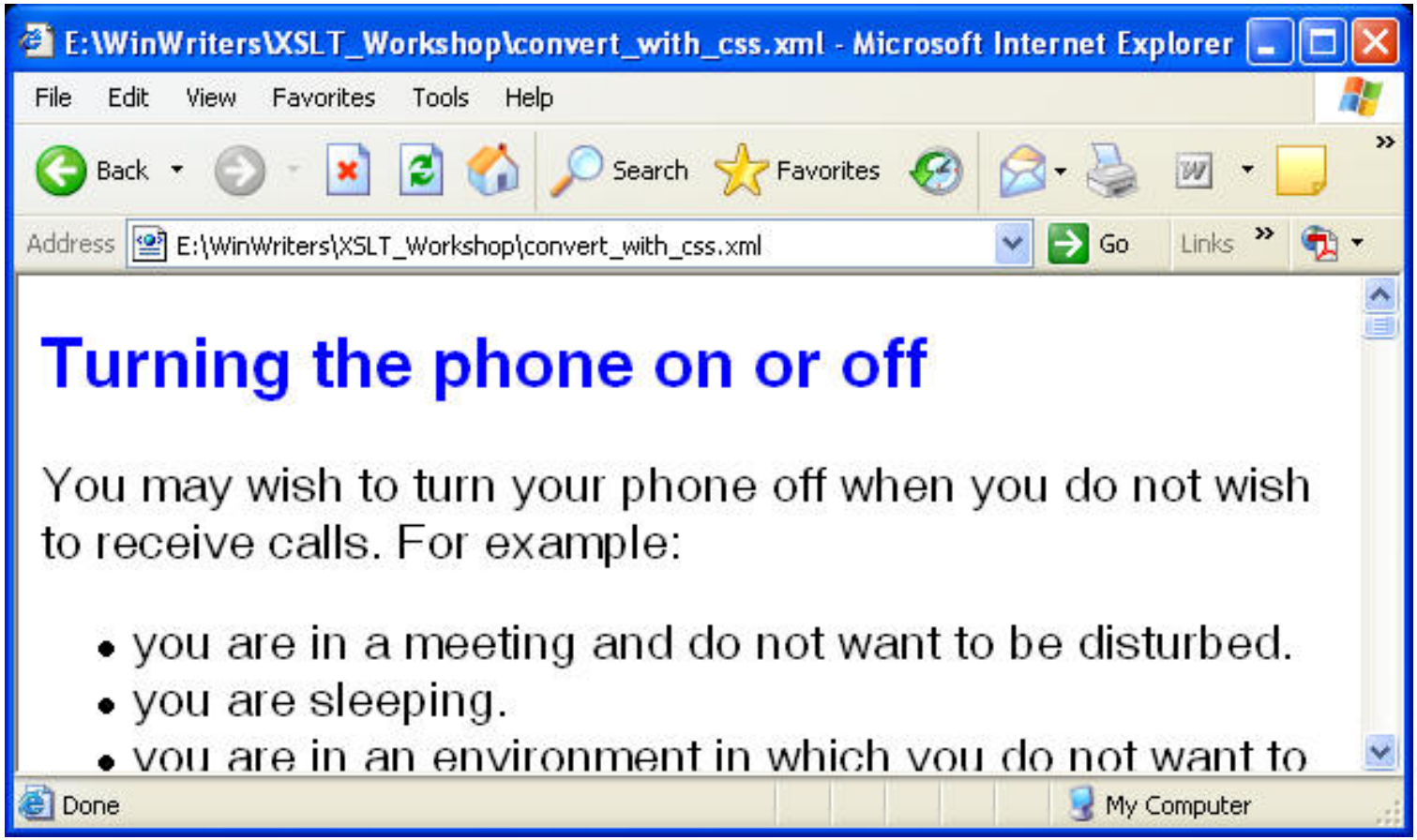
Modify `convert_with_css.xsl` to reference the CSS stylesheet “tasks.css” in the HTML result.

Open `convert_with_css.xml` in your browser to view your progress.

Experiment by modifying `tasks.css`. Refresh your browser to see any changes.

Exercise: Combining XSLT and CSS

Result of opening convert_with_css.xml in Internet Explorer.



Task: Filtering content from XML source

My XML document includes metadata that identifies particular categories of content.

- I want to suppress some categories in my result file.

or

- I want to include only specific categories in my result file.

Task: Filtering content from XML source

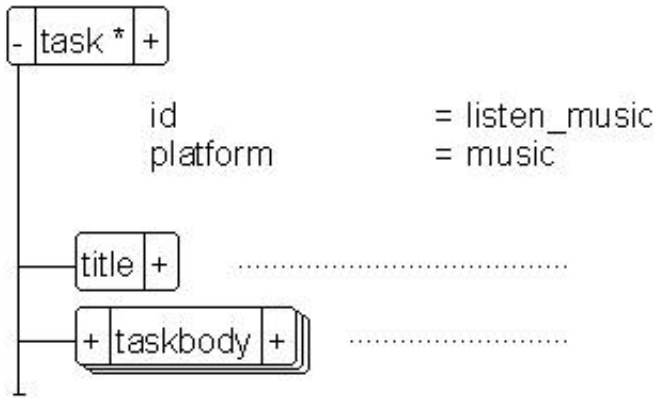
Scenario: Building documentation for a cell phone model that does not provide music support

Listening to music

You can use your phone to play MP3 music files. Use headphones with a 1/8-inch stereo jack.

To listen to music:

1. Plug your headphones into the headphone jack.
2. Press the **menu** key, and choose **music library**.
3. Click the **volume** buttons to adjust the music vol
4. Scroll through the list of songs with the arrow key to play the songs in your library, in order, until yo



Task: Filtering content from XML source

Create an “empty” template that matches the elements you want to suppress from your output.

```
<xsl:template match=“topic[@platform=‘music’]”/>
```

Be sure a more general template matches other instances of the element.

```
<xsl:template match=“topic">
   <xsl:apply-templates/>
</xsl:template>
```

Exercise: Filtering content from XML source

Modify filter_content.xsl to supress tasks with a platform value of “music” or “video”. Open filter_content.xml and verify that these tasks do not appear in the result.

Task: Adding named anchors for linking

To support HTML links, each potential target needs a named anchor. If your XML includes unique ID attributes, you can build the named anchors with the following template:

```
<xsl:template match="task" >
    <a name="{@ID}" />
    <xsl:apply-templates />
</xsl:template>
```

Task: Adding named anchors for linking

If link targets in your XML source (for example, heading elements) do not include unique ID attributes, you can use the generate-id() function to create unique named anchors for linking.

```
<xsl:template match="task" >
    <a name="{generate-id()}" />
    <xsl:apply-templates />
</xsl:template>
```

Exercise: Adding named anchors for linking

Modify named_anchors.xsl to add an HTML “named anchor” to each task. Open named_anchors.xml and verify that the named anchors appear in the HTML result (hint: use the right-click “View XSL Output” command).

Concept: XPath Expressions

Xpath is a companion language to XSLT. The two languages are separate W3C Recommendations, but are always used together.

Xpath includes *functions* that are useful for testing XML content (for example, comparing strings) and *expressions* for addressing parts of the XML tree.

Concept: XPath Expressions

Question: How do we create a link to another element in the XML tree?

Answer: Use XPath axes to address elements from the perspective of the “current” element. (Formally known as the “context node”).

Task: Adding Top and Bottom Links

We can add “go-to-top” and “go-to-bottom” navigation links by linking to the first and last topics. XPath provides a last() function that we can use for addressing the first and last topic.

```
<xsl:template  match=”topic” >
<tr>
   <td><a  href="#{generate-id(
   preceding::task[last()])}">First</a></td>
   <td><a  href="#{generate-id(
   following::task[last()])}">Last</a></td>
</tr>
</xsl:template>
```

Exercise: Adding Top and Bottom Links

Modify nav_links_top_bottom.xsl to add a “First” and “Last” link to each task. Open nav_links_top_bottom.xml and verify that the links work as expected.

Task: Adding Prev/Next Navigation Links

For each task, we can add links to the “Previous” and “Next” tasks.

```
<a href="#{generate-id(
   preceding::task[1])}">Prev</a></td>

<a href="#{generate-id(
   following::task[1])}">Next</a></td>
```

Exercise: Adding Prev/Next Navigation Links

Modify nav_links_sequence.xsl to add a “Prev” and “Next” link to each task. Open nav_links_sequence.xml and verify that the links work as expected.

Concept: Conditional Tests

XSLT supports conditional testing.

```
<xsl:if test="boolean condition">
Statements, literal elements
</xsl:if>
```

If boolean condition is true, content of <xsl:if> is executed.

Task: Conditionalize Navigation Links

A “Prev” navigation link is meaningless for the first topic. A “Next” navigation link is meaningless for the last topic. If this topic is first, don’t include a “Prev” link. If last, don’t include a “Next” link.

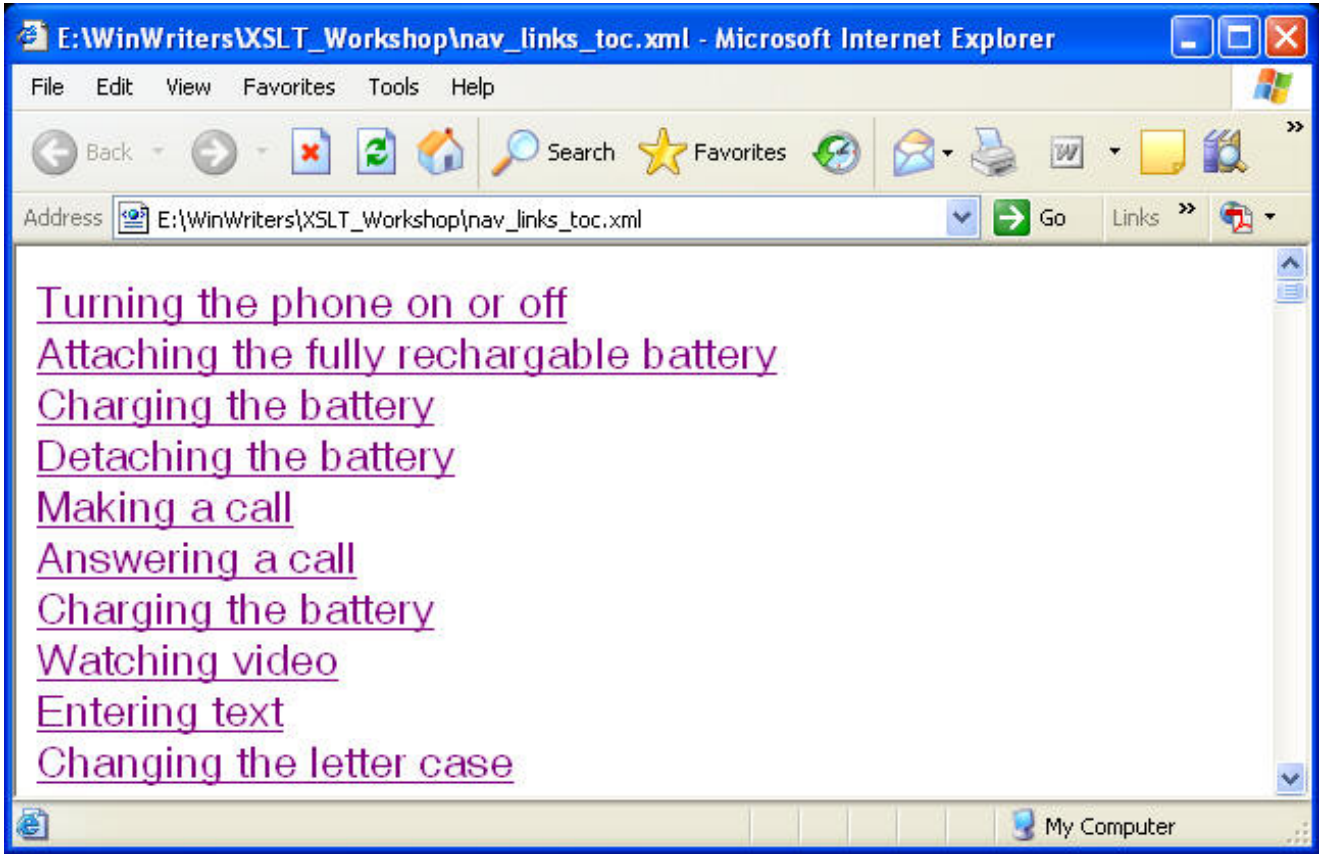
```
<xsl:if test="preceding::task[1]">
<a href="#{generate-id(preceding::task[1])}">Prev</a>
</xsl:if>
<xsl:if test="following::task[1]">
<a href="#{generate-id(following::task[1])}">Next</a>
</xsl:if>
```

Exercise: Conditionalize Navigation Links

Modify nav_links_conditional.xsl to check that a “Prev” or “Next” topic exists. Open nav_links_conditional.xml and verify that the first topic does not have a “Prev” link and that the last topic does not have a “Next” link.

Task: Adding TOC links

We can automatically generate TOC links from our XML content.



Concept:Iterating through an XML Tree

You can use <xsl:for-each > to iterate over a series of XML elements.

In template that sets up the HTML page, iterate through each task. Generate a link to each task using the generate-id() function.

```
<xsl:for-each  select="task"  >

<a href="#{generate-id(.)}">
     <xsl:value-of  select="title"/></a>

<br />

</xsl:for-each>
```

Task: Adding TOC links

In template that sets up the HTML page, iterate through each task. Generate a link to each task using the generate-id() function.

```
<xsl:for-each  select="task"  >

<a href="#{generate-id(.)}">
     <xsl:value-of  select="title"/></a>

<br />

</xsl:for-each>
```

Exercise: Adding TOC links

- Modify nav_links_toc.xsl to generate a list of topics at the top of the HTML page. Open nav_links_toc.xml to verify the result.
- Extra credit: Use <xsl:sort /> in the <xsl:for-each> element to sort the TOC links. Verify that the links work.
- Extra credit: Use <xsl:sort /> to sort the order of the tasks in the body of the HTML document.

Manipulating DITA Documents

- Unlike other XML vocabularies, DITA elements are not (should not be) manipulated by name, but by the value of the element's class attribute. This is an artifact of the DITA specialization mechanism.
- Matching class attribute values is a bit tricky, and yields more complex stylesheets.

Manipulating DITA Documents (2)

```
<step  class  ="- topic/li task/step ">

<!-- steps -->

</step>


<xsl:template match="*[contains(@class,'
task/step ')]”>

<!-- Process step -->

</xsl:template>
```

Contact Us!

We hope you enjoyed this presentation. Please feel free to contact us:

Alan Houser
arh@groupwellesley.com

Group Wellesley, Inc.
933 Wellesley Road
Pittsburgh, PA 15206
USA
412-450-0532
www.groupwellesley.com

tekom jahres tagung 2015
STUTTGART, 10.–12. NOV.

tcworld conference 2015
STUTTGART, NOVEMBER 10–12

Your opinion is important to us! Please tell us what you thought of the lecture. We look forward to your feedback via smartphone or tablet under

http://IN50.honestly.de

or scan the QR code

The feedback tool will be available even after the conference!