XSLT 3.0 Testbed

Tony Graham
Mentea
13 Kelly’s Bay Beach
Skerries, Co Dublin
Ireland
info@mentea.net
http://www.mentea.net

Version 1.0 – 15 February 2014
© 2014 Mentea
XSLT 3.0 Testbed

xsl:attribute-set vs map of functions 9
XHTML tables and xsl:iterator 11
Idioms 15
References 17
Appendix A – About 19
XSLT 3.0 testbed

What? How?
- Trying out new XSLT 3.0 features
- Converting existing JATS stylesheets to XSLT 3.0

Why?
- Early start on patterns and idioms to help adoption
- Find infelicities in spec (and implementations)
- The time is right
  - Project started November 2013
  - XSLT 3.0 Last Call WD – 12 December 2013

W3C Process
- End game for a W3C spec:
  - Last Call
  - Candidate Recommendation
  - Proposed Recommendation
  - Recommendation
- Changes after “Last Call” require more documentation and substantiation

What is JATS?
- Journal Article Tag Suite
- Successor to “National Library of Medicine (NLM) Journal Archiving and Interchange Tag Suite”
- Over 2 million articles in PubMed Central (in 2011)
- Used by journal publishers and archives around the world
- Nature announced two Open Access journals in January 2014
- Used by Public Library of Science (PLOS)
Why JATS?

- Simpler than, e.g., DocBook or TEI
- Not a toy
- Potentially useful to authors and archives
- Existing XSLT stylesheets available

JATSPreviewStylesheets

https://github.com/NCBITools/JATSPreviewStylesheets

- XSLT 1.0
  - Easy for new contributors to add XSLT 2.0-isms
- Public domain
  - No copyright issues
  - XSLT 3.0 stylesheets also public domain
  - Explicitly not supporting gazillion customisation parameters, PIs, etc.
- Simpler processing
- Fewer user expectations
Explicitly not supporting customisation

xslt3testbed goals

- Trial different techniques
- Develop patterns and idioms
- Develop XSLT 3.0 package for XHTML tables
  - `xsl:package` new in XSLT 3.0
  - XHTML tables used in many document types

xslt3testbed non-goals

- Single best way of doing anything
  - Multiple ways to solve the same problem are okay
- Definitive XSLT 3.0 testbed
  - It’s easy to fork and make your own version
- Complete stylesheet for all of JATS
  - Existing stylesheets don’t cover everything yet either

Results so far

- Trying out maps, anonymous functions, and `xsl:iterate`
- Small advances in multiple areas
- Not expecting to find a meteorite
6 W3C Bugzilla bug tickets so far

<table>
<thead>
<tr>
<th>ID</th>
<th>Product</th>
<th>Component</th>
<th>Assignee</th>
<th>Status</th>
<th>Resolution Summary</th>
<th>Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>XPath / XSLT 3.0</td>
<td>node</td>
<td>Jonatan Aguiu</td>
<td>NEW</td>
<td>nightly-level element and attribute constructors for use in dynamic functions</td>
<td>Mar 12, 2014</td>
</tr>
<tr>
<td>2002</td>
<td>XPath / XQuery 3.0</td>
<td>jonathan.roole</td>
<td>Asdf</td>
<td>ASSIGNED</td>
<td>XPath: the FunctionBuild production in body of spec</td>
<td>2014-03-31</td>
</tr>
<tr>
<td>2005</td>
<td>XPath / XQuery 3.0</td>
<td>jonathan.roole</td>
<td>REFO</td>
<td>POINT</td>
<td><code>use-namespace-uri</code> in <code>createElement</code> in XPath/Query 3.01</td>
<td>2014-03-07</td>
</tr>
<tr>
<td>2111</td>
<td>XPath / FunctionLib</td>
<td>REFO</td>
<td>FDE</td>
<td>WIP in draft</td>
<td>2013-02-01</td>
<td></td>
</tr>
<tr>
<td>2164</td>
<td>XPath / XSLT 3.0</td>
<td>REFO</td>
<td>FDE</td>
<td>use namespace URI in source position</td>
<td>2014-01-28</td>
<td></td>
</tr>
</tbody>
</table>

Other results

- One XSLT processor bug
- One change to Wendell Piez’s JATS Oxygen plug-in
- Technique for hosting Oxygen plugins on GitHub

4 JATSPreviewStylesheets patches so far

The JATSPreviewStylesheets network graph

Other results
\textbf{xsl:attribute-set vs map of functions} \hfill 16

- Compare \texttt{xsl:attribute-set} to map of functions
- Two tables with red table head text and blue table body text
- One with \texttt{<table style="orange">}

\begin{tabular}{|c|c|}
\hline
\textbf{1 head cell, 1 body cell.} \\
Head \\
Body \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
\textbf{I want to be orange.} \\
Head \\
Body \\
\hline
\end{tabular}

\section*{Stylesheets} \hfill 17

- \texttt{orange.xml} - Handles table[@style eq 'orange']
- \texttt{red-blue.xml} - Imports others
- \texttt{red.xml} - Red header text
- \texttt{blue.xml} - Blue body text
- \texttt{xhtml-tables.xml} - Base table stylesheet

\begin{tikzpicture}[node distance=2cm]
  \node (orange) {orange.xml};
  \node (red-blue) {red-blue.xml};
  \node (xhtml) {xhtml-tables.xml};
  \node (red) {red.xml};
  \node (blue) {blue.xml};
  \draw[->] (orange) -- (red-blue);
  \draw[->] (red-blue) -- (xhtml);
  \draw[->] (xhtml) -- (red);
  \draw[->] (xhtml) -- (blue);
\end{tikzpicture}

\section*{xsl:attribute-set} \hfill 18

\begin{Verbatim}
<\texttt{xsl:attribute-set} name="thead">
  <\texttt{xsl:attribute} name="color" select="'red'" />
</\texttt{xsl:attribute-set}>
\end{Verbatim}

- Since XSLT 1.0
- Generate named sets of attributes
- Evaluate \texttt{xsl:attribute} declaration in current context
- Multiple \texttt{xsl:attribute-set} with same name aggregate
- Refer to with \texttt{@use-attribute-sets} or \texttt{@xsl:use-attribute-sets}
**xsl:attribute-set has current context**

```
<xsl:attribute-set name="fig">
  <xsl:attribute name="id" select="generate-id()" /></xsl:attribute-set>
```

- Different value for every context
- Can also call templates and functions inside `xsl:attribute`

**xsl:attribute-set and the tables**

- Attribute set for each element type, e.g., `thead`
- Using *only* those attribute sets, can't do one table with orange style

<table>
<thead>
<tr>
<th>1 head cell, 1 body cell.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
</tr>
<tr>
<td>Body</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I want to be orange.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
</tr>
<tr>
<td>Body</td>
</tr>
</tbody>
</table>

**Map of functions**

- Element names as key, function to apply as value
- Context passed as function parameter

```
<xsl:template match="thead">
  <xsl:next-match>
    <xsl:with-param
      name="table-functions"
      as="map(xs:string, function(element()) as attribute()*)"
      select="map {
        'thead' := function($context as element()) as attribute()* [ 
          x3tb:attribute('color', 'red')
        ]
      }"
      tunnel="yes" />
  </xsl:next-match>
</xsl:template>
```

(_x3tb:attribute_() is XSLT function containing `xsl:attribute`)
Map of functions and the tables

- Import precedence and using `xsl:next-match` misses effect from `red.xsl`
- Could work around with this example, but harder in large system
- Possible work with map of sequence of functions or modifying of result of lower-precedence templates

```
1 head cell, 1 body cell.
Head
Body

I want to be orange.
Head
Body
```

`xsl:attribute-set` vs map of functions

- `xsl:attribute-set`:
  - Aggregating `xsl:attribute-set` across modules “just works”
  - Need other mechanism for per-element customisations
  - Can’t “turn off” an attribute, only override
- Map of functions:
  - Can do per-element overrides
  - More work for XSLT developer than `xsl:attribute-set`
  - Aggregating across modules and overriding templates needs more thought
  - Map of sequence of functions could be interesting
  - Is it just reinventing `typeswitch`?

XHTML tables and `xsl:iterator`

- Insert `<td/>` in gaps in table
- Consider rowspans
- Use `xsl:iterate`

<table>
<thead>
<tr>
<th>Head</th>
<th>1.1, 2.1</th>
<th>1.2</th>
<th>1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>2.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Head</th>
<th>1.1, 2.1</th>
<th>1.2</th>
<th>1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>2.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**xsl:iterate**

- Sequential processing of a sequence
- "many XSLT users find writing recursive functions to be a difficult skill, and this construct promises to be easier to learn"
- "[S]hould be more amenable to optimization"

**xsl:iterate**

```xsl
<xsl:iterate
    select = expression >
    <!-- Content: (xsl:param*, sequence-constructor,
    xsl:on-completion?) -->
</xsl:iterate>
```

- Parameters just like xsl:for-each/xsl:param

**xsl:on-completion**

```xsl
<xsl:on-completion
    select? = expression >
    <!-- Content: (sequence-constructor) -->
</xsl:on-completion>
```

- Last within xsl:iterate
- Additional (or only) result from xsl:iterate
**xsl:break, xsl:next-iteration**

```xml
<xsl:break
    select? = expression >
    <!-- Content: (sequence-constructor) -->
</xsl:break>
```

- “Last” within `xsl:iterate`
- Break out of `xsl:iterate`

```xml
<xsl:next-iteration>
    <!-- Content: (xsl:with-param*) -->
</xsl:next-iteration>
```

- “Last” within `xsl:iterate`
- Modify some or all parameters for next iteration
- If no `xsl:iterate`, parameters not changed

**When is “last”?**

- Last in `xsl:iterate`
- Last in last:
  - `xsl:if`
  - `xsl:when`
  - `xsl:otherwise`
  - `xsl:catch`
  - `xsl:try`
  - I.e., in “tail position” of element in tail position of ...
- So, conditionally set/change parameters in a “last” `xsl:if` or `xsl:choose`

**Applying xsl:iterate to table**

- `xsl:iterate` over `<tr>` in `<thead>, <tbody>, <tfoot>`
- Inner `xsl:iterate` over `<th>` and `<td>` in `<tr>`
- Keep map of columns and remaining rowspan for the column
- Update map in inner `xsl:iterate` for each `<th>` or `<td>`
- Emit `<td/>` if more columns than cells
**xsl:iterate in action**

```xml
<xsl:template match="thead | tbody | tfoot">
  <xsl:param name="col-map" as="map(xs:integer, element(col))" />
  <xsl:variable name="cell-map" select="map:new(for $i in 1 to count(map:keys($col-map)) return map:entry($i, 0))"
    as="map(xs:integer, xs:integer)" />
  <xsl:copy>
    <xsl:apply-templates select="@*" mode="table-copy"/>
    <xsl:iterate select="tr">
      <xsl:param name="cell-map" select="$cell-map" as="map(xs:integer, xs:integer)" />
      <xsl:message select="('row', count(map:keys($cell-map)))"/>
      <xsl:variable name="content" as="item()+">
        <xsl:iterate select="1 to count(map:keys($cell-map))">
          <xsl:param name="colnum" select="1" as="xs:integer"/>
          <xsl:param name="cell-map" select="$cell-map" as="map(xs:integer, xs:integer)" />
          <xsl:param name="cells" select="*" as="element()*"/>
          <xsl:message select="map:get($cell-map, .)"/>
          <xsl:choose>
            <xsl:when test="map:get($cell-map, .) > 0">
              <xsl:message select="'rowspanned'"/>
              <xsl:next-iteration>
                <xsl:with-param name="colnum" select="$colnum + 1" as="xs:integer"/>
                <xsl:with-param name="cell-map" select="map:new(($cell-map,
                  map:entry(., map:get($cell-map) - 1)))" as="map(xs:integer, xs:integer)"/>
              </xsl:next-iteration>
            </xsl:when>
            <xsl:when test="empty($cells)">
              <xsl:message select="'no cell'"/>
              <td/>
              <xsl:next-iteration>
                <xsl:with-param name="colnum" select="$colnum + 1"/>
              </xsl:next-iteration>
            </xsl:when>
          </xsl:when>
        </xsl:iterate>
        <td/>
        ...
      </xsl:message>
    </xsl:message>
  </xsl:iterate>
</xsl:template>
```

**Code, continued**

```xml
<xsl:choose>
  <xsl:when test="map:get($cell-map, .) > 0">
    <xsl:message select="'rowspanned'"/>
    <xsl:next-iteration>
      <xsl:with-param name="colnum" select="$colnum + 1" as="xs:integer"/>
      <xsl:with-param name="cell-map" select="map:new($cell-map,
        map:entry(., map:get($cell-map, .) - 1))" as="map(xs:integer, xs:integer)"/>
    </xsl:next-iteration>
  </xsl:when>
  <xsl:when test="empty($cells)">
    <xsl:message select="'no cell'"/>
    <td/>
    <xsl:next-iteration>
      <xsl:with-param name="colnum" select="$colnum + 1"/>
    </xsl:next-iteration>
  </xsl:when>
</xsl:choose>
```
Code, continued

```xml
<xsl:otherwise>
  <xsl:message select="concat('cell: ''', $cells[1]), '''; rowspan: ', string($cells[1]/@rowspan)"
  <xsl:apply-templates select="$cells[1]" />
  <xsl:next-iteration>
    <xsl:with-param name="colnum" select="$colnum + 1" as="xs:integer" />
    <xsl:with-param name="cell-map" select="map:new((($cell-map,
      map:entry(., xs:integer($cells[1]/@rowspan, 1)[1]) - 1)))" />
    <xsl:with-param name="cells" select="$cells[position() > 1]" />
  </xsl:next-iteration>
</xsl:otherwise>
</xsl:choose>
</xsl:iterate>
</xsl:variable>
```

Code, continued

```xml
<xsl:copy>
  <xsl:apply-templates select="@*" mode="table-copy" />
  <xsl:sequence select="$content[position() != last()]" />
</xsl:copy>
<xsl:next-iteration>
  <xsl:with-param name="cell-map" select="$content[last()]" />
</xsl:next-iteration>
</xsl:iterate>
</xsl:template>
```

**xsl:iterate issues**

- Less readable (IMO) than recursive templates or functions
- Returning map in `xsl:on-completion` of inner `xsl:iterate`
  - Buffering result of `<tr>` just to use `last()` to get map
  - Maybe split map calculation and `<tr>` processing or use `xsl:accumulator`?
- Not sure why there’s no `xsl:sort` with `xsl:iterate`

**Idioms**

- “manner of expression characteristic of or peculiar to a language"
- Every language, human or computer, has them
- Different in different versions of XSLT
XSLT 1.0 – Meunchian grouping

- Developed by Steve Meunch
- Because XSLT 1.0 didn't do grouping ... now forgotten?

```xml
<xsl:key name="contacts-by-surname" match="contact" use="surname" />
<xsl:template match="records">
  <xsl:for-each select="contact[key('contacts-by-surname', surname)[1] = 1]">
    <xsl:sort select="surname" />
    <xsl:value-of select="surname" />,<br />
    <xsl:for-each select="key('contacts-by-surname', surname)">
      <xsl:sort select="forename" />
      <xsl:value-of select="forename" /> ( <xsl:value-of select="title" /> )<br />
    </xsl:for-each>
  </xsl:for-each>
</xsl:template>
```

XSLT 2.0 – `xsl:choose` shortcuts

- XSLT 1.0
  ```xml
  <xsl:attribute name="text-align">
    <xsl:choose>
      <xsl:when test="@align">@align</xsl:when>
      <xsl:otherwise>from-table-column()</xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
  </fo:table-cell>
  ```

- XPath 2.0 if
  ```xml
  <xsl:attribute name="text-align">"if (exists(@align))
  then @align
  else 'from-table-column()'"</xsl:attribute>
  </fo:table-cell>
  ```

- XPath 2.0 sequence constructor
  ```xml
  <xsl:attribute name="text-align">"[@align, 'from-table-column()'][1]]"</xsl:attribute>
  </fo:table-cell>
  ```

XSLT 3.0 – ???

Predicting...

- Common, safe streaming constructs
- Higher-order functions and named function references
  - XSLT 2.0:
    ```xml
    if (@firstpage eq 0)
    then floor($totalPagesDecimal)
    else ceiling($totalPagesDecimal)
    ```
  - XSLT 3.0 (and validating processor only?):
    ```xml
    (floor#, ceiling#)[@firstpage + 1]($totalPagesDecimal)
    ```
Conclusion

https://github.com/MenteaXML/xslt3testbed

- The time is right
- Useful in multiple arenas
- Go fork and multiply

References

- slide 4 – W3C Process Document
  http://www.w3.org/2005/10/Process-20051014/tr.html
- slide 5 – NISO Z39.96 The Journal Article Tag Suite (JATS): What Happened to the NLM DTDs?
- slide 13 – Bugs so far
  https://www.w3.org/Bugs/Public/buglist.cgi?email1=tgraham%40mentea.net&emailreporter1=1&emailtype1=substring&product=XPath%20%20XQuery%20%20XSLT&query_format=advanced
- slide 18 – xsl:attribute-set
  http://www.w3.org/TR/xslt-30/#attribute-sets

References

- slide 26 – xsl:iterate
  http://www.w3.org/TR/xslt-30/#iterate
- slide 37 – idiom
  http://www.thefreedictionary.com/idiom
- slide 40 – Functional XPath commands
Appendix A

About

Tony Graham 19
Mentea 19

Tony Graham

Tony Graham has been working with markup since 1991, with XML since 1996, and with XSLT/XSL-FO since 1998. He is Chair of the Print and Page Layout Community Group at the W3C and previously an invited expert on the W3C XML Print and Page Layout Working Group (XPPL) defining the XSL-FO specification, as well as an acknowledged expert in XSLT, developer of the open source xmlroff XSL formatter, a committer to both the XSpec and Juxy XSLT testing frameworks, the author of “Unicode: A Primer”, a member of the XML Guild, and a qualified trainer.

Tony’s career in XML and SGML spans Japan, USA, UK, and Ireland, working with data in English, Chinese, Japanese, and Korean, and with academic, automotive, publishing, software, and telecommunications applications. He has also spoken about XML, XSLT, XSL-FO, EPUB, and related technologies to clients and conferences in North America, Europe, and Australia.

Mentea

Mentea specialises in consulting and training in XML, XSL-FO, & XSLT. We are available for on-site meetings and classes, worldwide, but as well as on-site meetings and classes, we routinely keep in touch with clients though email, Skype, instant messaging, and telephone and through a secure, per-client or per-project wiki, revision-control, and issue-tracking system.

Our staff have been working with markup since 1991, with XML since 1996, and with XSLT/XSL-FO since 1998. Based in Dublin, Ireland, Mentea has a global reach: in recent projects, we have helped companies and organisations in the USA, Ireland, England, and France with their XSLT, XSL, and XML, including:

• Writing Schematron for a professional body
• Augmenting a XSLT-based automated schema documentation system that produces both HTML and PDF
• Extending FOP for a software company
• Training in XML, oXygen, DocBook, XSLT 2.0, and XSL-FO
• Formatting JATS to PDF for a scientific journal
• Writing XSLT stylesheets to convert non-XML into XML then into EPUB
• Writing XSLT to convert Excel into XML for a commercial bank

Mentea presents a unique range of skills extending beyond XML and XSL-FO/XSLT into Unicode, SGML, DSSSL, and programming in C, Java, Perl, Lisp, and other languages.

We understand how markup works. Our staff has worked with markup in Japan, USA, UK, and Ireland as user, consultant, and developer, with data in English, French, Chinese, Japanese, and Korean, with academic, automotive, publishing, software, and telecommunications applications, and in the Web Services and document processing arenas.

We are also interested in applying the tools for ensuring software quality – unit testing, code coverage, profiling, and other tools – to XML and XSLT/XSL-FO processing.

Through our associations and affiliations with other consultants around the world, we can call on extra help for large or specialised projects.