XML-in-Practice 2009

A Conference of IDEAlliance

SEPTEMBER 30 - OCTOBER 1
ARLINGTON HILTON
WASHINGTON DC METRO
ARLINGTON, VIRGINIA USA

PRODUCED BY

IDEAlliance
International Digital Enterprise Alliance ®

XML-in-Practice 2009

A Conference of IDEAlliance

Upgrading Stylesheets from XSLT 1.0 to 2.0

Priscilla Walmsley
Managing Director, Datypic
http://www.datypic.com
pwalmsley@datypic.com

PRODUCED BY

IDEAlliance
International Digital Enterprise Alliance ®

Agenda

- Status/process of upgrading XSLT to 2.0
- Backward compatibility issues
- Tips for improving XSLT using 2.0 features

The Status of XSLT 2.0

- W3C Recommendation since January 2007
- Multiple processors now available:
  - Saxon
    - open source version that handles both XSLT 2.0 and XQuery.
    - commercial version that provides advanced features such as schema support and more sophisticated optimization
  - AltovaXML
    - free XSLT 2.0 engine with schema support
  - IBM
    - commercial version (WebSphere feature pack)
  - More on the way...
    - Intel, others?

Why Upgrade to 2.0?

- Great new features
  - Grouping
  - Regular expression support
  - Multiple result documents
  - Convenient enhancements to XPath (if-then-else, simple for loop)
  - Schema and typing support
  - Many smaller conveniences (tunnel parameters, multiple modes, etc.)
- Desire to use an XSLT 2.0 processor
  - Need to ensure that existing stylesheets are still interpreted the same way when using a 2.0 processor

Upgrading XSLT to Version 2.0

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
```

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
```

...and we're done!

(just kidding)

Backward Compatibility Issues

Upgrading Gotchas

- The bad news:
  - There are some backward incompatibilities between XSLT 1.0 and 2.0
- The good news:
  - There are not that many common cases, and they are well documented in the W3C recommendations
  - A 1.0 stylesheet can used with a 2.0 processor
    - Any XPath 1.0 expression is also a syntactically valid XPath 2.0 expression
    - Any XSLT 1.0 stylesheet is also a syntactically valid XSLT 2.0 stylesheet
  - XPath 1.0 backward compatibility mode can ease the transition

XPath 1.0 Compatibility Mode

- Allows you to treat certain backward-incompatible XPath expressions as they were treated in version 1.0
- You can add a "version" attribute to any XSLT 2.0 element
  - turns on compatibility mode for that element and all its descendents

```
<xsl:template match="book" version="1.0">
      <xsl:text>Author: </xsl:text>
      <xsl:value-of select="author"/>
</xsl:template>
```

- You can even add it at the xsl:stylesheet level, which means 1.0 compatibility mode is in effect throughout

Data Model Differences in 2.0

- Nodes
  - root nodes are now "document nodes"
  - namespace nodes are deprecated
- Atomic values
  - more types, e.g. **`xs:integer, xs:date`**
- Sequences (formerly node-sets)
  - are ordered
  - can contain duplicates
  - can contain atomic values as well as nodes

Handling Multiple Items

- Some instructions and functions in XPath/XSLT 1.0 choose the first node if given a set of more than one
- Equivalent XPath/XSLT 2.0 instructions/functions either use all of the items or raise an error
  - Uses all the values:
    - The select attribute of xsl:value-of
    - The select attribute of xsl:attribute
    - Attribute value templates ("{" and "}" in attribute values)
    - The value attribute of xsl:number
  - Raises an error:
    - The select attribute of xsl:sort
    - Built-in functions that expect a single item

Handling Multiple Items
Example 1: xsl:value-of

```
<books>
    <book>
      <author>Walmsley</author>
      <title>Definitive XML Schema</title>
    </book>
    <book>
      <author>Goldfarb</author>
      <author>Walmsley</author>
      <title>XML in Office 2003</title>
    </book>
</books>
```

```
<xsl:template match="book">
    <xsl:text>Author: </xsl:text>
    <xsl:value-of select="author"/>
</xsl:template>
```

1.0 Output	2.0 Output
Author: Walmsley Author: Goldfarb	Author: Walmsley Author: Goldfarb **Walmsley**

Handling Multiple Items
Example 2: Attribute Value Templates

```
<books>
    <book>
      <author>Walmsley</author>
      <title>Definitive XML Schema</title>
    </book>
    <book>
      <author>Goldfarb</author>
      <author>Walmsley</author>
      <title>XML in Office 2003</title>
    </book>
</books>
```

```
<xsl:template match="book">
    <newBook author="{author}"/>
</xsl:template>
```

1.0 Output

```
<newBook author="Walmsley"/>
<newBook author="Goldfarb "/>
```

2.0 Output

```
<newBook author="Walmsley"/>
<newBook author="Goldfarb
    Walmsley"/>
```

Handling Multiple Items
Example 3: xsl:sort

```
<books>
    <book>
      <author>Walmsley</author>
      <title>Definitive XML Schema</title>
    </book>
    <book>
      <author>Goldfarb</author>
      <author>Walmsley</author>
      <title>XML in Office 2003</title>
    </book>
</books>
```

```
<xsl:template match="books">
    <xsl:for-each select="book">
      <xsl:sort select="author"/>
      Author: <xsl:value-of select="author"/>
    </xsl:for-each>
</xsl:template>
```

1.0 Output

Author: Goldfarb
Author: Walmsley

2.0 Output

Error XTTE1020

Handling Multiple Items
Example 4: substring Function

```
<books>
    <book>
      <author>Walmsley</author>
      <title>Definitive XML Schema</title>
    </book>
    <book>
      <author>Goldfarb</author>
      <author>Walmsley</author>
      <title>XML in Office 2003</title>
    </book>
</books>
```

```
<xsl:template match="book">
    <xsl:text>Author: </xsl:text>
    <xsl:value-of
    select="substring(author,1,32)"/>
</xsl:template>
```

1.0 Output	2.0 Output
Author: Walmsley Author: Goldfarb	*Error XPTY0004*

Handling Multiple Values Safely

- To create an expression that will work in both 1.0 and 2.0, the best approach is to put a "[1]" on the end of the path expression:
  - <xsl:value-of select="author[1]"/>
  - <newBook author="{author[1]}"/>
  - <xsl:sort select="author[1]"/>
  - <xsl:value-of select="substring(author[1],1,32)"/>

Type Sensitivity

- 2.0 is more strongly typed than 1.0
- The type system is based on XML Schema
  - built-in types from XML Schema (e.g. xs:integer, xs:string, xs:date) are automatically built into the language
  - other types can be imported from a schema
- Values of a certain type can be constructed
  - e.g. xs:date("2004-12-15")

Strong Typing in XPath/XSLT 2.0

- Expressions return values of a certain type:
  - the current-date() always returns an xs:date value
  - a subtraction (using the minus sign) will always return a number
- Certain operators expect the values to be of a certain type:
  - e.g., a minus sign expects the arguments to be numbers
  - the substring function expects the first argument to be a string
- Some type conversion happens automatically
  - if values are untyped (come from a schema-less document), they are automatically cast to the appropriate types
  - if values are typed (come from a document with a schema, or are the result of another expression that returns a specific type) they are *not* automatically cast, and will raise an error

Do I Have to Pay Attention to Types?

- Usually, no, not if you don't want to.
  - without a schema, your XML data is "untyped"
  - in most expressions, untyped values are cast to the expected types

product/price + 23

sum(product/price)

if price is untyped, it is cast to a numeric type automatically

- Sometimes, yes, if your value does have a specific type

substring-after(product/price,'$') + 23

value has been turned into a string before being added; this will raise an error in 2.0

Type Sensitivity
Example 1: Arithmetic

```
<product>
    <title>Some product</title>
    <price>$12.50</price>
    <discount>$2.50</discount>
</product>
```

```
<xsl:template match="product">
    Final price: <xsl:value-of select="substring-after(price,'$') -
                                       substring-after(discount,'$')"/>
</xsl:template>
```

1.0 Output

Final price: 10

2.0 Output

Error XPTY0004, because minus operator does not support strings as arguments

Remedy: call the number function:

```
<xsl:value-of select="number(substring-after(price,'$')) - number(substring-
    after(discount,'$'))"/>
```

Comparisons and Types

- In 2.0, comparisons (=, <, etc.) can be used on atomic values of non-numeric types
  - strings
  - dates, times, durations
- Comparing an untyped value to a typed value is straightforward:
  - The expression "price > 5" casts the price element into a number
  - The expression "name = 'foo' " casts the name element into a string
- However, two untyped values are treated like *strings*
  - in 1.0, arguments in a comparison were assumed to be numbers

Type Sensitivity
Example 2: Comparing 2 Typed Values

```
<items>
    <item id="ID001">One item</item>
    <item id="ID002">Another item</item>
    <item id="ID104">A third item</item>
    <item id="ID105">A fourth item</item>
</items>
```

```
<xsl:template match="item">
    <xsl:if test="substring-after(@id,'ID') > 100">
        Name: <xsl:value-of select="."/>
    </xsl:if>
</xsl:template>
```

1.0 Output	2.0 Output
Name: A third item Name: A fourth item	*Error XPTY0004, because you cann ot compare a string to an integer*

Type Sensitivity
Example 3: Comparing Untyped and Typed Values

```
<items>
    <item id="001">One item</item>
    <item id="002">Another item</item>
    <item id="104">A third item</item>
    <item id="105">A fourth item</item>
</items>
```

```
<xsl:template match="item">
    <xsl:if test="@id > 100">
      Name: <xsl:value-of select="."/>
    </xsl:if>
</xsl:template>
```

1.0 Output	2.0 Output
Name: A third item Name: A fourth item	Name: A third item Name: A fourth item

Type Sensitivity
Example 4: Comparing 2 Untyped Values

```
<products>
    <product id="1">
      <discount>10.00</discount>
      <price>12.99</price>
    </product>
    <product id="2">
      <discount>9.00</discount>
      <price>12.99</price>
    </product>
</products>
```

```
<xsl:template match="product">
    <xsl:value-of select="@id"/>
    <xsl:value-of select="discount > price"/>
    <xsl:value-of select="discount > 5"/>
</xsl:template>
```

1.0 Output

```
1  false true
2  false true
```

2.0 Output

```
1  false true
2  true true
```

Fatal vs. Recoverable Errors

- Processors were allowed to ignore (or recover from) certain error conditions in 1.0, but not in 2.0:
  - Passing a parameter to a template which did not have a parameter by that name defined
  - Trying to create an element or attribute using xsl:element/xsl:attribute with an invalid XML name
  - Specifying a "mode" or "priority" attribute on a template with no "match" attribute
  - Generating attributes of an element after children of that element

Improving XSLT Using 2.0 Features

Grouping

- New **`xsl:for-each-group`** element allows you to iterate through groups
  - **`select`** attribute identifies items to group
  - **`group-by`** attribute specifies the grouping key
- Two functions can be used within **`for-each-group`**:
  - **`current-group()`** returns members of current group
  - **`current-grouping-key()`** returns the current grouping key

Grouping by Value Input and Output

```
<Songs>
  <Song>
    <Title>Help!</Title>
    <Artist>McCartney</Artist>
  </Song>
  <Song>
    <Title>Helter Skelter</Title>
    <Artist>Lennon</Artist>
  </Song>
  <Song>
    <Title>Hey Jude</Title>
    <Artist>Lennon</Artist>
  </Song>
  <Song>
    <Title>Love Me Do</Title>
    <Artist>McCartney</Artist>
  </Song>
</Songs>
```

Input document

```
<Songs>
    <Artist name="McCartney">
            <Song>Love Me Do</Song>
            <Song>Help!</Song>
    </Artist>
    <Artist name="Lennon">
            <Song>Helter Skelter</Song>
            <Song>Hey Jude</Song>
    </Artist>  ...
</Songs>
```

Desired output document

Grouping
The Old Way (Muenchian Grouping)

```
<xsl:key name="songs-by-artist" match="Song" use="Artist" />
<xsl:template match="Songs">
  <Songs>
    <xsl:for-each select="Song[generate-id() =
      generate-id(key('songs-by-artist', Artist)[1])]">
      <Artist name="{Artist}">
        <xsl:apply-templates select=
          "key('songs-by-artist', Artist)"/>
      </Artist>
    </xsl:for-each>
  </Songs>
</xsl:template>
<xsl:template match="Song">
  <Song><xsl:value-of select="Title"/></Song>
</xsl:template>
```

Grouping
The New Way (xsl:for-each-group)

```
<xsl:template match="Songs">
  <Songs>
  <xsl:for-each-group select="Song"  group-by="Artist" >
   <Artist name="{current-grouping-key()}">
        <xsl:apply-templates select= "current-group()"/>
   </Artist>
  </xsl:for-each-group>
  </Songs>
</xsl:template>
<xsl:template match="Song">
  <Song><xsl:value-of select="Title"/></Song>
</xsl:template>
```

Grouping by Sequence Inputs and Outputs

```
<body>
 <h1>Chapter 1</h1>
 <p>This chapter...</p>
 <h2>Section 1.1</h2>
 <p>In this section...</p>
 <p>More text</p>
 <h2>Section 1.2</h2>
 <p>In this section...</p>
</body>
```

Input document

```
<section level="1">
  <section level="2">
   <heading>Chapter 1</heading>
   <p>This chapter...</p>
  </section>
  <section level="2">
   <heading>Section 1.1</heading>
   <p>In this section...</p>
   <p>More text</p>
  </section>
  <section level="2">
   <heading>Section 1.2</heading>
   <p>In this section...</p>
  </section>
</section>
```

Desired output document

Grouping by Sequence Example

```
<xsl:template match="/" >
  <xsl:for-each-group select="body/*" group-starting-with="h1" >
   <section level="1" >
    <xsl:for-each-group select="current-group()"
                        group-starting-with="h2" >
     <section level="2" >
      <xsl:apply-templates select="current-group()" />
     </section>
    </xsl:for-each-group>
   </section>
  </xsl:for-each-group>
</xsl:template>
```

Syntactic Conveniences
Using XPath to Return Atomic Values

- The last step in a path can now return an atomic value, not just a node

```
<xsl:for-each select=".//name">
  <xsl:value-of select="substring(.,1,32)"/>
</xsl:for-each>
```

In 1.0

```
<xsl:value-of select=".//name/substring(.,1,32)"/>
```

In 2.0

Syntactic Conveniences
Using the `string-join` Function

- Use the **`string-join`** function instead of looping through strings

```
<xsl:for-each select="//name">
  <xsl:value-of select="."/>
  <xsl:if test="position() != last()">,  </xsl:if>
</xsl:for-each>
```

In 1.0

```
<xsl:value-of select="string-join(//name,', ')"/>
```

In 2.0

Syntactic Conveniences
Using Range Expressions

- <*integer*> to <*integer*> syntax used to construct a sequence of consecutive integers
- Useful to iterate a specific number of times

recursive templates that add a count on each call, or iterating through fake nodes

In 1.0

```
<xsl:for-each select="1 to 5">
      ...
</xsl:for-each>
```

In 2.0

Syntactic Conveniences
Conditional Expressions

```
<xsl:variable name="heading">
  <xsl:choose>
    <xsl:when test="desc">
      <xsl:value-of select="substring(desc,1,32)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="name"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

In 1.0

```
<xsl:variable name="heading"
select="if  (desc)
          then substring(desc,1,32)
          else name"/>
```

In 2.0

Eliminating Extensions

- Many EXSLT, processor-specific and custom extensions can be eliminated when upgrading to 2.0
- Many new built-in functions in 2.0 (115 or so)
  - date and time functions
  - regular expressions
  - some new math functions and operators
- Eliminating extensions makes sense if you are committing wholly to 2.0
  - makes your XSLT more portable
  - could improve performance

Replacing EXSLT for Current Date/Time

```
<xsl:stylesheet    xmlns:date="http://exslt.org/dates-and-times"
   version="1.0">
...
  <xsl:variable name="now"  select="date:date-time()"/>
...
```

In 1.0

```
<xsl:stylesheet
   version="2.0">
...
  <xsl:variable name="now"   select="current-dateTime()"/>
...
```

In 2.0

Replacing EXSLT node-set()

```
<xsl:variable name="devstatuses">
   <m val="DD" disp="Dev. Delay or Disability"/>
   <m val="AR" disp="At-Risk"/> ...
</xsl:variable>
<select name="devStatus">
  <xsl:for-each select="exslt:node-set($devstatuses)/*">
    <option value="{@val}"> <xsl:value-of select="@disp"/> </option>
  </xsl:for-each>
</select>
```

In 1.0

```
<xsl:variable name="devstatuses">
   <m val="DD" disp="Dev. Delay or Disability"/>
   <m val="AR" disp="At-Risk"/> ...
 </xsl:variable>
<select name="devStatus">
  <xsl:for-each select="$devstatuses/*">
    <option value="{@val}"><xsl:value-of select="@disp"/> </option>
  </xsl:for-each>
</select>
```

In 2.0

Introducing Schemas in XSLT 2.0

- Schemas can pass type information to the stylesheet
- Use of schemas is *optional*
- You can:
  - validate input and/or result documents
  - import schema documents, which lets you:
    - know when your XSLT violates the schema
    - do special processing based on types

Using Schemas to Catch Static Errors

```
<xsl:stylesheet .....
<xsl:import-schema
  namespace="http://www.datypic.com/prod"
  schema-location="prod.xsd" />

<xsl:template match="schema-element(catalog)">
  <xsl:for-each select="produt">
    <xsl:sort select="product/number"/>
    <xsl:value-of select="name + 1"/>
  </xsl:for-each>
</xsl:template>
```

misspelling

invalid path; product will never have product child

type error: name is declared to be of type xs:string, so cannot be used in an add operation

XSLT 2.0 Resources

- XSLT 2.0 Recommendation:
  - http://www.w3.org/TR/xslt20
- Reference Materials:
  - Marston, David. *Planning to Upgrade XSLT 1.0 to 2.0*, 7-part series, developerWorks.com.
  - Kay, Michael. *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wrox, 2008.
- Reusable XSLT 2.0 functions:
  - http://www.xsltfunctions.com

XML-in-Practice 2009
A Conference of IDEAlliance

Thank you for your interest

Feel free to contact me at:

Email: pwalmsley@datypic.com
Website: http://www.datypic.com

PRODUCED BY
IDEAlliance
International Digital Enterprise Alliance ®