XSLT 2.0: More power, less hassle

Doug Tidwell
IBM Corporation

dtidwell@us.ibm.com
Our agenda

- A brief history of XSLT
- Changes in XPath 2.0
- Parameters, variables and functions
- Grouping
- Working with multiple documents
- Regular expressions
- Extending XSLT
- Sample transformations
A brief history of XSLT
XSLT and XPath 1.0

- XSLT 1.0 and XPath 1.0 became official recommendations of the W3C on November 16, 1999.
  - See w3.org/TR/xslt and w3.org/TR/xpath
  - Both specs printed on letter-sized paper: 139 pages.
XSLT 2.0 + XPath 2.0

- XSLT 2.0 and XPath 2.0 are defined in eight documents:
  - XSLT 2.0 – w3.org/TR/xslt20/
  - XPath 2.0 – w3.org/TR/xpath20/
  - XSLT 2.0 and XQuery 1.0 Serialization
    - Defines how the data structures created by XSLT stylesheets and XQueries can be written to various formats.
      - w3.org/TR/xslt-xquery-serialization/
More XSLT 2.0 and XPath 2.0 specs:

- XQuery 1.0 and XPath 2.0 Functions and Operators
  - Defines language keywords, functions and operators
    - [w3.org/TR/xpath-functions/](http://w3.org/TR/xpath-functions/)

- XQuery 1.0 and XPath 2.0 Data Model
  - Defines how XQuery and XPath work with data
    - [w3.org/TR/xpath-datamodel/](http://w3.org/TR/xpath-datamodel/)
XSLT 2.0 + XPath 2.0

More XSLT 2.0 and XPath 2.0 specs:

- XQuery 1.0 and XPath 2.0 Formal Semantics
  - Provides a formal definition of everything in XQuery 1.0 and XPath 2.0.
  - Fans of EBNF diagrams rejoice!
  - w3.org/TR/xquery-semantics/
XQuery 1.0

- The XQuery specs are considered part of the package.
  - XQuery 1.0
    - w3.org/TR/xquery
  - XML Syntax for XQuery (XQueryX)
    - w3.org/TR/xqueryx
XSLT 2.0 + XPath 2.0

- All these specs together are roughly 1,100 letter-sized pages.
- XSLT 2.0 and XPath 2.0 also use XML Schema, defined in three more documents at the W3C:
  - Primer – w3.org/TR/xmlschema-0/
  - Structures – w3.org/TR/xmlschema-1/
  - Datatypes – w3.org/TR/xmlschema-2/
What's new?

Here are some of the biggest changes:

- Everything is a *sequence* (no more nodesets).
- There are now *atomic values*.
- XPath 2.0 features powerful new operators and keywords that simplify your stylesheets (less recursion)
- Grouping is much easier.
What's new?

- Another major change is *XML Schema support*.
- This allows you to do very complex validation of your documents as you create them or transform them.
What's new?

- Schema support adds a great deal of complexity to the language.
- Currently the only schema-aware XSLT 2.0 processor is a closed-source, commercial product.
Validation

- **Validation is expensive.**
  - Your goal should be to validate an XML document once and only once.
  - In the database world, we push validation down into the database; we don't validate database constraints in every client application.
  - Over a pint or two, we can discuss whether validation in a stylesheet is useful...
Changes in XPath 2.0
New in XPath 2.0

- Everything is a sequence
- Datatypes
- New operators: if, to, some, every, for, idiv, etc.
- Casting: instance of, cast as, castable as, treat as
- Dates, times and durations
- 17,382 new functions
Sequences

- In XPath 2.0, everything is a **sequence**. A sequence can contain nodes or **atomic values**.
  - An atomic value is something like 3 or 'Guinness' or [the value returned by the function call] `true()`.
- The specs also mention a **singleton**; a singleton is a sequence of length 1.
Sequences

Here’s a sequence:

```xml
<xsl:variable name="lotteryNumbers" as="xs:integer*"
    select="'(1, 3, 8, 14, 21, 42)'"/>
```
Datatypes

- `xs:string`
- `xs:boolean`
- `xs:float`
- `xs:double`
- `xs:decimal`
- `xs:integer`
- `xs:anyURI`
- `xs:QName`
- `xs:hexBinary, xs:base64Binary`
- `xs:date`
- `xs:time`
- `xs:dateTime`
- `xs:duration`
- `xs:gDay, xs:gMonth, xs:gYear, xs:gMonthDay, xs:gYearMonth`
- `xs:NOTATION`
Other datatypes

There are five other datatypes defined by XPath 2.0:

- `xs:untyped`
- `xs:untypedAtomic`
- `xs:anyAtomicType`
- `xs:dayTimeDuration`
- `xs:yearMonthDuration`
XPath 2.0 operators

- []
- /, //
- unary – and +
- cast as
- castable as
- treat as
- instance of
- except, intersect
- |, union
- *, div, idiv, mod
- to
- =, !=, <, <=, >, >=
- is, eq, ne, gt, ge, lt, le
- <<, >>
- and
- or
- if, then, else
- some, every
- for
- , [comma]
XPath 2.0 operators

- A fine point:
  - The operators `=, !=, <, <=, >` and `>=` compare nodes and sequences.
  - The operators `lt, le, gt, ge, eq` and `ne` compare atomic values.
XPath 2.0 operators

- XPath 2.0 adds the **if**, **then** and **else** operators that can take the place of `<xsl:if>` and `<xsl:choose>`.
XPath 2.0 operators

- `<xsl:value-of select="if ($x gt 7) then 'big' else 'small'"/>

- This outputs the atomic string values **big** or **small**, but we don't have to use `<xsl:if>` or `<xsl:choose>` to generate them.

  ➢ This also uses the XPath 2.0 **gt** operator to compare atomic values.
Schema-aware processors

- If you're using a *schema-aware* XSLT processor, you can use your own data types.
  - Currently there are no open source XSLT processors that are schema-aware (AFAIK).
**XPath expressions**

- **FLWOR expressions:**
  - `fn:sum(for $i in (/purchase-order/items) return (item/price * item/qty))`
  - This `for` expression replaces the recursive technique you'd have to use in XSLT 1.0.

- FLWOR is `for`, `let`, `where`, `order` and `return`.
XPath expressions

- Other types of expressions:
  - Datatypes: `instance of`, `cast as`, `castable as`, `treat as`
  - Sets: `except`, `intersect`, `union`
  - Ranges and iterators: `to`, `for`
  - Constructors: `xs:integer(3)`
  - (:: Comments ::)
Constructors

- You can use the name of a data type as a constructor function.
  
  ```xml
  <xsl:variable name="a" select="xs:integer(3)"/>
  ```

- You can also use the `as` attribute to specify a variable's data type.
  
  ```xml
  <xsl:variable name="a" select="'3'" as="xs:integer"/>
  ```
Outputting values

- In XSLT 2.0, the `<xsl:value-of>` element has the `separator` attribute.

```xml
<xsl:value-of select="1 to 10" separator="", "/>
```

writes this text to the output:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Outputting values

- To do the same thing in XSLT 1.0, you had to do something like this:

```xml
<xsl:for-each select="...">
  <xsl:value-of select="."/>
  <xsl:if test="not(position()=last())">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:for-each>
```
Numbering things

- XSLT 2.0 adds some new features to the `<xsl:number>` element:
  - `format="w"` produces the numbers one, two, three, ...
  - `format="W"` produces the numbers ONE, TWO, THREE, ...
  - `format="Ww"` produces the numbers One, Two, Three, ...
Numbering things

- The **ordinal** attribute is new.
  Numbering things with `ordinal="yes"` gives these results:
  - With `format="1"` – 1st, 2nd, 3rd, ...
  - With `format="w"` – first, second, third, ...

Doug Tidwell – XSLT 2.0: More Power, Less Hassle
Numbering things

- The `lang` attribute works with these new features.
  - `format="w:"` and `lang="de"` generates `eins: , zwei: , drei: ,`
  - `format="Ww: "` and `lang="de"` and `ordinal="yes"` generates `Erste: , Zweite: , Dritte: ,`

- Not all combinations will be supported by your processor...
Backwards behavior

- XSLT 2.0 lets you put the `version` attribute on any XSLT element. This lets you use XSLT 1.0 behavior on a given element if you want.
  - For example, in XSLT 1.0, `1 div 0` returned `Infinity`; in XSLT 2.0, the stylesheet won't run at all.
  - If you want the XSLT 1.0 behavior, you can add `version="1.0"` to the element.
A quick and confusing note about math in XSLT 2.0:

- Dividing an `xs:float` or `xs:double` by zero returns `INF` (infinity).
- Dividing anything else by zero (`xs:integer`, `xs:decimal`, `xs:yearMonthDuration` or `xs:dayTimeDuration`) is a fatal error.
Dates & times

- The new functions `format-date()`, `format-time()` and `format-dateTime()` let you format dates and times.
- To use these functions, you have to pass arguments that have the correct datatype.
Dates & times

- Here are some examples:

  ```xml
  <xsl:value-of select="format-date(fn:current-date(),
    '[D01]/[M01]/[Y0001]') />
  returns 22/10/2007.

  format-time(fn:current-time(),
    '[h1]:[m01] [P]')
  returns 8:24 a.m.
  ```
Dates & times

- Another example:
  
  \[
  \text{format-dateTime(fn:dateTime())}, \ ' [H01]: [M01] \ [z] \ on \ [D]/[M]' \)
  \]
  returns 08:24 GMT on 22/10.

- Notice that these examples use the new \text{fn:current-date()}, \text{fn:current-time()} and \text{fn:current-dateTime()} functions.
Parameters, variables and functions
Parameters

- XSLT 2.0 makes several changes to parameters.
- Passing extra parameters with `<xsl:call-template>` is a static error.
  - Confusingly enough, passing extra parameters with `<xsl:apply-templates>`, `<xsl:apply-imports>` or `<xsl:next-match>` is *not* an error.
Required parameters

- In XSLT 2.0 you can specify that a parameter is required:
  
  `<xsl:param name="x" required="yes"/>

- If a parameter is required, it cannot have a default value.
  
  In other words, a parameter with a required attribute can't have a select attribute, and it can't have content.
Tunnel parameters

- XSLT 2.0 introduces the concept of *tunnel parameters*.
  - These are similar to dynamically-scoped variables in some functional programming languages.
- We'll use a short example to show how these are used and where they're useful...
Tunnel parameters

```xml
<xsl:apply-templates
    select="*|text()">
  <xsl:with-param
      name="font-size" select="20"/>
</xsl:apply-templates>
```
Tunnel parameters

- This passes the parameter `font-size` to whatever templates are invoked.
  - We have to do this because some template down the line might need it.
  - As other templates invoke each other, they have to pass this parameter also.
  - Every time we create a new template, we have to pass this parameter.
Tunnel parameters

- A tunnel parameter is silently passed along as one template invokes another.

```xml
<xsl:apply-templates
    select="*|text()">
  <xsl:with-param
    name="font-size" select="20"
    tunnel="yes"/>
</xsl:apply-templates>
```
Tunnel parameters

- Other templates can use the tunnel parameter whenever they want:

  `<xsl:template match="code">
    <xsl:param name="font-size" tunnel="yes"/>
  </xsl:template>`

  The parameter is now visible.

- If we code `<xsl:param name="font-size"/>`, the XSLT processor assumes this is a new parameter.
Variables

- In XSLT 2.0, the `as` attribute lets you define the datatype of a variable.

  ```xml
  <xsl:variable name="age" as="xs:integer" select="'41'"/>
  <xsl:variable name="members" as="xs:string[*"
                          select="address/name"/>
  ```

- Notice that the variable `$members` is a sequence, not an atomic value.
Functions

- XSLT 2.0 introduces the `<xsl:function>` element. It lets you create what's essentially a named template that returns a value.
- We'll look at an example next...
Functions

```xml
<xsl:variable name="colors" as="xs:string*">
  select="('red','blue','green')"/>
<xsl:function name="sample:getColor" as="xs:string">
  <xsl:param name="pos" as="xs:integer"/>
  <xsl:value-of
  select="$colors[$pos]"/>
</xsl:function>
```
Functions

- The `as` attribute defines the return type of the function.
  - In this example, our function returns a single string.
  - If the function returned a sequence of strings, we would use `as="xs:string*"` instead.

- Notice that we gave our function a namespace; that separates it from other functions.
Functions

- Another benefit of `<xsl:function>` is that we can call our functions like any other functions:
  ```xml
  <td bgcolor="{sample:getColor(1)}">
  ```
- Keep in mind that the first position in a sequence is `[1]`, not `[0]`. 
Functions

- A final note: We could have (should have) written the `<xsl:value-of>` element like this:
  
  ```
  <xsl:value-of select="$colors[(pos mod count($colors)) + 1]"/>
  ```

- Our original code, `colors[pos]`, gives us a runtime error if `pos` is larger than the size of the sequence `colors`. 
Grouping
Grouping

- For an example, we'll take some addresses and group them by state. We want a listing like this:
  - **Customers in Massachusetts:**
    - Mary Backstayge, Skunk Haven
    - Harry Backstayge, Skunk Haven
    - Amanda Reckonwith, Lynn
  - **Customers in Maine:**
    - Natalie Attired, Winter Harbor
  - …
Grouping in XSLT 1.0

...sucks.
Grouping in XSLT 2.0

- XSLT 2.0 introduces the `<xsl:for-each-group>` element, which *greatly* simplifies grouping.
Grouping in XSLT 2.0

There are three basic steps:

1. Define an XPath expression for the property we'll use for grouping. *We don't need a key.*
2. Use an XPath expression to select all of the nodes we want to group.
3. Iterate through the unique grouping values with `<xsl:for-each-group>`.
Grouping in XSLT 2.0

- There are two useful functions you can use inside `<xsl:for-each-group>`:
  - `current-group()` returns the sequence of nodes that match the current grouping key.
  - `current-grouping-key()` returns the current grouping key.
Grouping in XSLT 2.0

- There are four different grouping styles:
  - group-by
  - group-adjacent
  - group-starting-with
  - group-ending-with

- We'll look at each of these next.
This is the most common grouping style.
The example we just went through with the Muench method can be handled very easily with **group-by**.
group-by

<xsl:for-each-group
    select="/addressbook/address"
    group-by="state">
    <xsl:sort select="state"/>
    <xsl:for-each
        select="current-group()">[Handle each item in the group]<xsl:for-each>
</xsl:for-each-group>
In this example, `<xsl:for-each-group>` uses `group-by` to create the groups. Whenever we find a value of `<state>` we haven't seen before, a new group is created. Otherwise, the current address is added to an existing group.
group-by

- The `<xsl:sort select="state">` element sorts the groups.
  - Without this element, the first group would be based on the first `<state>` element in the document. The second group would be based on the `<state>` element with the second unique value, etc.
**group-adjacent**

- **group-adjacent** is useful when we want to group items because they're next to each other.
  
  ➢ We'll convert a set of adjacent paragraphs into an unordered list, with each paragraph converted into a list item.
<for-each-group
  select="html/body/*"
  group-adjacent="if (self::p) then true else false">
The `group-adjacent` attribute returns a value.

- That value can be any atomic type (integer, boolean, date, whatever)
- The only thing the XSLT processor cares about is whether this node's value matches the previous node's value.
Our example had `group-adjacent` return one of two values, but there's no limit on what `group-adjacent` can return:

```
group-adjacent="if (self::*p[@class='item']) then 1 else if (self::*p[@class='note']) then 2 else 3"
```
**group-starting-with**

- **group-starting-with** defines what starts a group.
  - When the XSLT processor finds this, it starts a new group.
  - Everything goes into that group until the processor finds another [whatever] that starts a new group.
group-starting-with

<xsl:for-each-group select="*"
group-starting-with="h1">
  <sect1>
    <xsl:apply-templates
      select="current-group()"/>
  </sect1>
</xsl:for-each-group>
**group-starting-with**

- **Note:** The XSLT processor creates a new group starting with the first selected item, *whether it matches the start condition or not.*
  
  In this example, if the first item is a `<p>` element, it starts a new group. Whenever the XSLT processor finds an `<h1>` element, it starts another group.
group-ending-with

- This variation lets you specify the condition that ends the current group.
  - This is most often used when putting data into columns.

```xml
<for-each-group
  group-ending-with="position() mod 3 = 0">
  Every third item closes the current group.
</for-each-group>
```
Finding unique values

- Finding all the unique values in a set of values is a common task (this is part of grouping, for example).
- XPath 2.0 has the delightfully simple `fn:distinct-values()` function.
  - Given a sequence, it returns a sequence of unique values.
- To do the same thing in XSLT 1.0, you have to use recursion.
A final note

A final note on sorting and grouping:

*Remember that you might have more efficient tools for sorting and grouping.*

- If your XML data comes from a relational database, use the SQL `ORDER BY` and `GROUP BY` statements when you select it.
Creating multiple output documents
Multiple output documents

- Most XSLT 1.0 processors provided an extension element to create multiple output documents.
  - Xalan had the `<redirect:write>` element,
  - Saxon had the `<saxon:output>` element,
  - and so forth.
- XSLT 2.0 provides the same function with the `<xsl:result-document>` element.
Result document

- This creates a separate output document:

```xml
<xsl:result-document method="html" href="toc.html">
  <html>
    <h1>Table of contents</h1>
    
    ... 

  </html>
</xsl:result-document>
```
Result document

- The `href` attribute defines the name of the output document. You can create the file name with an AVT:
  
  ```
  href="{concat('chapter', position(), '.html')}"
  ```

- The `method` attribute supports `xml`, `html`, `xhtml` and `text`.
  
  ➢ It's the same as the `method` attribute of `<xsl:output>`. 
Regular expressions
Regular expressions

- As you'd expect from any modern language, XSLT 2.0 includes support for regular expressions.
- There are three new elements:
  - `<xsl:analyze-string>`
  - `<xsl:matching-substring>`
  - `<xsl:non-matching-substring>`
Regular expressions

- The `<xsl:analyze-string>` uses a `select` attribute to define what text should be analyzed, and it has a `regex` attribute to specify the regular expression.

  `<xsl:analyze-string select="." regex="([0-9]{3})">"
Regular expressions

- Notice the syntax within the regular expression. Curly braces are used to define attribute value templates, so we use double braces ({{ and }}) for the regular expression.
  - {{3}} means "three occurrences," while {3} is evaluated to the number 3.
Regular expressions

- The `<xsl:analyze-string>` element can contain three things:
  - A single `<xsl:matching-substring>`
  - A single `<xsl:non-matching-substring>`
Regular expressions

Here's a more complicated example:

```xml
<xsl:analyze-string select="." regex="([0-9]{3})-([0-9]{4})">
  <xsl:matching-substring>
    <xsl:text>This is a valid phone number, extension #</xsl:text>
    <xsl:value-of select="regex-group(1)"/>
  </xsl:matching-substring>
</xsl:analyze-string>
```
Regular expressions

- More of our example:

```xml
<xsl:non-matching-substring>
  <xsl:text>Bad phone number!</xsl:text>
</xsl:non-matching-substring>
</xsl:analyze-string>
```
Regular expressions

- Using `<xsl:analyze-string>` with 555-1738 generates these results:
  This is a valid phone number, extension #555
Regular expressions

- Using `<xsl:analyze-string>` with `5552-3829` generates these results:

  Bad phone number!
Regular expressions

- The XSLT 2.0 `regex-group()` function returns the section of the analyzed string that matches a parenthesized section of the regular expression.
Regular expressions

- Given the text 123 and the regular expression
  
  `(\[0-9\]\{1\}) (\[0-9\]\{2\})`:

  - `regex-group(1)` returns 1
  - `regex-group(2)` returns 23
Regular expressions

- The `<xsl:analyze-string>` element also has a `flags` attribute, which lets you control the way the regular expression is evaluated.

  ➢ You can make the regular expression case-insensitive and control how newline and whitespace characters are processed, for example.
Regular expressions

- Flags:
  - `s` – The dot character (.) matches any character, including the newline character (\&#xA;).
  - `m` – Regular expressions are evaluated in multiline mode.
    - Normally `^` matches the start of the entire string and `$` matches the end of the entire string. In multiline mode, `^` matches the start of any line, and `$` matches the end of any line.
Regular expressions

- More flags:
  - `i` – Regular expressions are case-insensitive
  - `x` – All whitespace characters (`\x9;`, `\xA;`, `\xD;` and `\x20;`) are removed from the regular expression before it is evaluated.

- You can use the flags in any order; `flags="ixs"` and `flags="six"` are the same.
Regular expressions

- Finally, there are other functions that help you work with regular expressions:
  - matches()
  - replace()
  - tokenize()

- All of these functions use the flags parameter as well.
Regular expressions

- The `matches()` function returns true or false, depending on whether a string matches a regular expression.
  - `fn:matches("abracadabra", "bra")` returns true, for example.
  - `fn:matches("abracadabra", "BRA", "i")` returns true as well.
Regular expressions

- The `replace()` function lets you replace the portion of a string that matches a regular expression with another string.
  ```
  replace("abracadabra", "bra", "*")
  ```
  returns "a*cada*", for example.
Regular expressions

- The `tokenize()` function breaks a string into tokens, using a regular expression as the separator.

  ```
  tokenize("The cat sat on the mat", \s+) returns the sequence ("The", "cat", "sat", "on", "the", "mat").
  ```
Extending XSLT
Extending XSLT

- XSLT 1.0 and 2.0 provide an extension mechanism.
  - You can create extension elements and extension functions.
- These typically tie you to a particular XSLT implementation...
- The new functions added in XSLT 2.0 and XPath 2.0 remove many of the cases where you need extensions.
Extending XSLT

- We’ll look at some XSLT 2.0 extensions that handle language-specific sorting and string comparisons.
- For sorting, we’ll sort the Spanish word *llaves* after *luna*.
- For string comparisons, we’ll say the German words *Strasse* and *Straße* are the same.
Extension elements

- An extension element gets access to the XSLT processor's context. It can use the variables, output streams, and anything else available to the processor.

- Saxon features extension elements such as `<sql:connect>` and `<sql:query>` for working with SQL databases.
Sample transformations
Sample transformations

- We’ll wrap up with a look at several transformations from XML to various formats.
A brief ethics test
Is it ethical...

...to mention the second edition of my book is on its way? (ISBN 0-596-51415-8)

Lists for $49.99, but it’s a bargain at twice the price.

Rough Cut version available today!

No! No!
Summary
Summary

- XSLT is an elegant, XML-based language for transforming XML documents into something else.
- *I hope you’re convinced that XSLT 2.0 is a powerful language for transforming documents.*
Thanks!

Doug Tidwell
IBM Corporation
dtidwell@us.ibm.com