Improve your XSLT 2.0 stylesheets with types and schemas

Specify schema types, parameters types, and return values with XSLT 2.0 for easier debugging and maintenance

Priscilla Walmsley

May 15, 2012

The type-aware and schema-aware features of XSLT 2.0 can greatly assist you when you debug a stylesheet, and improve stylesheet quality and robustness in handling all input data. Learn how to use type-aware and schema-aware XSLT 2.0 during the debugging and testing process to avoid common issues with invalid paths, incorrect assumptions about data types, and cardinalities. Also, find examples of XSLT stylesheets that contain errors that would not be caught if schema-aware features were not in use, and discover how explicitly specifying types results in useful error messages.

XML content can be complex and unpredictable. When you process it using XSLT 1.0, a lot of trial and error can be involved in defining the right expressions and handling all possible content structures. Misspelled names in XPath 1.0 expressions, for example, return nothing rather than providing a useful error message, making them difficult to debug.

Frequently used acronyms

- HTML: Hypertext Markup Language
- XHTML: Extensible HTML
- XML: Extensible Markup Language
- XSLT: Extensible Stylesheet Language Transformations

XSLT 2.0 is a big improvement over version 1.0, and strong typing and schema awareness are useful added features. Declaring types for values in an XSLT stylesheet allows the processor to tell you when you make incorrect assumptions about the data type of the XML content or the number of occurrences of a particular element or attribute. This functionality results in much more useful error messages. Importing XML Schemas takes it a step farther, giving the processor enough information about the input document structure to inform you when you have invalid XPaths rather than simply return nothing. They also provide information about data types in the content, preventing you from performing operations that don't make sense for that data type.
Using types in XSLT

Most programming languages offer a way to specify the type of a variable or parameter. In XSLT 2.0, you can declare the types of expressions using the `as` attribute, which can appear in a number of places:

- On the `xsl:variable` or `xsl:param` element to indicate the type of that variable or parameter
- On the `xsl:template` or `xsl:function` element to indicate the return type of that template or function
- On the `xsl:sequence` element to indicate the type of the sequence
- On the `xsl:with-param` element to indicate the type of a value passed to a template

The value of the `as` attribute is known as a *sequence type*.

Using XML Schema built-in types

A common sequence type you might use in an `as` attribute is the name of a particular data type, such as string or integer. In XSLT, you would use one of the data types that are built into the XML Schema specification. To do so, you prefix the type name with, for example, `xs:` and declare the `xs:` namespace at the top of your stylesheet. *Table 1* lists the most commonly used XML Schema data types.

<table>
<thead>
<tr>
<th>Data type name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>Any text string</td>
<td>abc, this is a string</td>
</tr>
<tr>
<td>integer</td>
<td>An integer of any size</td>
<td>1, 2</td>
</tr>
<tr>
<td>decimal</td>
<td>A decimal number</td>
<td>1.2, 5.0</td>
</tr>
<tr>
<td>double</td>
<td>A double-precision floating-point number</td>
<td>1.2, 5.0</td>
</tr>
<tr>
<td>date</td>
<td>A date in YYYY-MM-DD format</td>
<td>2009-12-25</td>
</tr>
<tr>
<td>time</td>
<td>A time in HH:MM:SS format</td>
<td>12:05:04</td>
</tr>
<tr>
<td>boolean</td>
<td>A true/false value</td>
<td>true, false</td>
</tr>
<tr>
<td>anyAtomicType</td>
<td>A value of any of the simple types</td>
<td>a string, 123, false, 2009-12-25</td>
</tr>
</tbody>
</table>

For example, if I want to declare that the value of a variable is a date, I can use:

```
<variable name="effDate" select="bill/effDate" as="xs:date"/>
```

Simple values like strings and dates are known as *atomic values*. Note that if `effDate` is an element, its contents will be converted to an atomic value of type `xs:date` (assuming that no schema is associated with the input document).

Sequence types representing XML nodes

You can also use the more generic sequence types listed in *Table 2* to represent kinds of nodes in an XML tree. You don't prefix these sequence types with `xs:`, because they are not XML Schema types.
### Table 2. Sequence types representing XML nodes

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>element()</td>
<td>Any element</td>
</tr>
<tr>
<td>element(book)</td>
<td>Any element named book</td>
</tr>
<tr>
<td>attribute()</td>
<td>Any attribute</td>
</tr>
<tr>
<td>attribute(isbn)</td>
<td>Any attribute named isbn</td>
</tr>
<tr>
<td>text()</td>
<td>Any text node</td>
</tr>
<tr>
<td>node()</td>
<td>A node of any kind (element, attribute, text node, and so on)</td>
</tr>
<tr>
<td>item()</td>
<td>Either a node of any kind or an atomic value of any kind (for example, a string, integer)</td>
</tr>
</tbody>
</table>

For example, if I want to express that the value bound to a variable is an element, I can say:

```xml
<variable name="effDate" select="//effDate" as="element()"/>
```

Unlike in the previous example, this will not be converted to an atomic value; the variable will contain the element itself.

### Using occurrence indicators

You can also use the occurrence indicators listed in Table 3 to express how many of a particular item may appear. These occurrence indicators appear at the end of the sequence type, after the expression from Table 1 or Table 2.

### Table 3. Using occurrence indicators

<table>
<thead>
<tr>
<th>Occurrence indicator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Zero to many</td>
</tr>
<tr>
<td>?</td>
<td>Zero to one</td>
</tr>
<tr>
<td>+</td>
<td>One to many</td>
</tr>
<tr>
<td>(no occurrence indicator)</td>
<td>One and only one</td>
</tr>
</tbody>
</table>

For example, if I want to express that the value bound to a variable is zero to many elements, I can say:

```xml
<variable name="effDate" select="//effDate" as="element()"*/>
```

Zero elements (or zero items of any kind) is also known as the empty sequence.

### Using types to make your stylesheets more robust

You might wonder how the use of as attributes improves your stylesheets. Look at a couple of examples. You can download all of the code examples in this article as a .zip file (see Download).

### Scenario 1: Using types to enforce cardinalities

Listing 1 shows an input document that contains two books.
Listing 1. Sample book input (books.xml)

```xml
<books xmlns="http://www.datypic.com/books/ns">
  <book>
    <title>The East-West House: Noguchi's Childhood in Japan</title>
    <author>
      <person>
        <first>Christy</first>
        <last>Hale</last>
      </person>
    </author>
    <price>9.95</price>
    <discount>1.95</discount>
  </book>
  <book>
    <title>Buckminster Fuller and Isamu Noguchi: Best of Friends</title>
    <author>
      <person>
        <first>Shoji</first>
        <last>Sadao</last>
      </person>
    </author>
    <price>65.00</price>
    <discount>5.00</discount>
  </book>
</books>
```

Listing 2 is a stylesheet that transforms book input into HTML.

Listing 2. Scenario 1 XSLT, original version (xslt_before.xsl)

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dtyf="http://www.datypic.com/functions"
  xpath-default-namespace="http://www.datypic.com/books/ns">
  <xsl:output method="html"/>
  <xsl:template match="books">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="dtyf:format-person-name(author/person)"/></td>
      <td><xsl:value-of select="price"/></td>
    </tr>
  </xsl:template>
  <xsl:function name="dtyf:format-person-name">
    <xsl:param name="person"/>
    <xsl:value-of select="$person/last"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="$person/first"/>
  </xsl:function>
</xsl:stylesheet>
```
Using the provided input document, the XSLT runs as expected, giving the results shown in Figure 1.

**Figure 1. Results of original Scenario 1 XSLT on books.xml**

<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The East-West House: Noguchi's Childhood in Japan</td>
<td>Vitra Design Museum</td>
<td>$125.00</td>
</tr>
<tr>
<td>Buckminster Fuller and Isamu Noguchi: Best of Friends</td>
<td>Sadao, Shoji</td>
<td>$65.00</td>
</tr>
</tbody>
</table>

Suppose I try to run it on more varied test data, such as in morebooks.xml (Listing 3).

**Listing 3. More varied input document (morebooks.xml)**

```xml
<books xmlns="http://www.datypic.com/books/ns">
  <book>
    <title>Isamu Noguchi: Sculptural Design</title>
    <author>
      <institution>Vitra Design Museum</institution>
    </author>
    <price>$125.00</price>
  </book>
  <book>
    <title>The East-West House: Noguchi's Childhood in Japan</title>
    <author>
      <first>Christy</first>
      <last>Hale</last>
    </author>
    <price>$9.95</price>
    <discount>$1.95</discount>
  </book>
  <book>
    <title>Buckminster Fuller and Isamu Noguchi: Best of Friends</title>
    <author>
      <first>Shoji</first>
      <last>Sadao</last>
    </author>
    <price>$65.00</price>
    <discount>$5.00</discount>
  </book>
  <book>
    <title>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</title>
    <author>
      <first>Louise</first>
      <middle>Allison</middle>
      <last>Cort</last>
    </author>
  </book>
  <book>
    <title>Sculptural Design</title>
    <author>
      <first>Bert</first>
      <last>Winther-Tamaki</last>
    </author>
  </book>
  <book>
    <title>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</title>
    <author>
      <first>Bruce</first>
      <middle>J.</middle>
      <last>Altshuler</last>
    </author>
  </book>
</books>
```
Figure 2 shows that it has incomplete results. For books that have an institution as an author, it is missing. For those books with multiple authors, it puts all the first names before the comma and all the last names after it. Instead of providing me with error messages, the stylesheet silently gives me the wrong output.

**Figure 2. Results of original Scenario 1 XSLT on morebooks.xml**

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isamu Noguchi: Sculptural Design</td>
<td></td>
<td>125.00</td>
</tr>
<tr>
<td>The East-West House: Noguchi's Childhood in Japan</td>
<td>Hale, Christy</td>
<td>9.95</td>
</tr>
<tr>
<td>Buckminster Fuller and Isamu Noguchi: Best of Friends</td>
<td>Sadao, Shoji</td>
<td>65.00</td>
</tr>
<tr>
<td>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</td>
<td>Cort Winther-Tamaki Altshuler Niimi, Louise Bert Bruce Ru</td>
<td>29.95</td>
</tr>
</tbody>
</table>

To create a more robust stylesheet, I can amend the function as shown in **Listing 4**, where I add as attributes to both the parameter and the function (as a return type), making explicit the assumptions I made when I wrote the function.

**Listing 4. Scenario 1 XSLT, revised function**

```xsl
<xsl:function name="dtyf:format-person-name" as="xs:string*”
    <xsl:param name="person" as="element()”/>
    <xsl:value-of select="$person/last”/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="$person/first”/>
</xsl:function>
```

The sequence type `element()` indicates that one and only one element is allowed. When I run the revised XSLT, I get appropriate error messages:

- **An empty sequence is not allowed as the first argument of dtyf:format-person-name()** (for an institutional author)
- **A sequence of more than one item is not allowed as the first argument of dtyf:format-person-name()** (for multiple authors)

Based on these error messages, I know I need to change the XSLT as shown in **Listing 5** to handle both institutional authors and more than one author.

**Listing 5. Scenario 1 XSLT, final version**

```xsl
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform”
```
My results, in Figure 3, look a lot better and now show the institutional author and multiple authors correctly.

**Figure 3. Results of final Scenario 1 XSLT on morebooks.xml**

<table>
<thead>
<tr>
<th>Title</th>
<th>Museum</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isamu Noguchi: Sculptural Design</td>
<td>Vitra Design Museum</td>
<td>125.00</td>
</tr>
<tr>
<td>The East-West House: Noguchi's Childhood in Japan</td>
<td>Hale, Christy</td>
<td>9.95</td>
</tr>
<tr>
<td>Buckminster Fuller and Isamu Noguchi: Best of Friends</td>
<td>Sadao, Shoji</td>
<td>65.00</td>
</tr>
<tr>
<td>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</td>
<td>Cort, Louise Winther-Tamaki, Bert Altshuler, Bruce Naiti, Ryu</td>
<td>29.95</td>
</tr>
</tbody>
</table>
The return type of the function in this example doesn't do much, but it's good practice anyway to be explicit about what you are returning. In other cases, the results of a function will be used in an operation that is expecting a result of a certain type.

**Scenario 2: Using types to ensure correct interpretation of operators**

Continuing with the book example, Listing 6 shows an XSLT that now takes into account the discount element when calculating the price.

**Listing 6. Scenario 2 XSLT, original version**

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dtyf="http://www.datypic.com/functions"
xpath-default-namespace="http://www.datypic.com/books/ns">
  <xsl:output method="html"/>
  <xsl:template match="books">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
      <td><xsl:value-of select="dtyf:calculate-price(price, discount)"/></td>
    </tr>
  </xsl:template>
  <xsl:function name="dtyf:calculate-price">
    <xsl:param name="price"/>
    <xsl:param name="discount"/>
    <xsl:sequence select="$price - $discount"/>
  </xsl:function>
</xsl:stylesheet>
```

Again, this works fine on books.xml (except for a rounding error), but when I run it against morebooks.xml, it doesn't handle books that have no discount, as in Figure 4, because the value of $discount is an empty sequence, and any arithmetic operation on the empty sequence also returns the empty sequence.
I consider just comparing $discount to 0 to fix this, but then I decide that the function will be even more robust if I make sure the discount is less than the price. I don't want any negative prices in my table. So, I also include the comparison $discount < $price, as in Listing 7.

**Listing 7. Scenario 2 XSLT, revised version**

```xml
<xsl:function name="dtyf:calculate-price">
  <xsl:param name="price"/>
  <xsl:param name="discount"/>
  <xsl:choose>
    <xsl:when test="$discount > 0 and $discount < $price">
      <xsl:sequence select="$price - $discount"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:sequence select="$price"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>
```

The results in Figure 5 look good at first glance. The first book with the missing discount is right. But luckily, I look more closely and notice that the discount is not being subtracted for the last book.

**Figure 5. Results of revised Scenario 2 XSLT on morebooks.xml**

<table>
<thead>
<tr>
<th>Isamu Noguchi: Sculptural Design</th>
<th>Vitra Design Museum</th>
<th>125.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>The East-West House: Noguchi's Childhood in Japan</td>
<td>Christy Hale</td>
<td>7.999999999999999</td>
</tr>
<tr>
<td>Buckminster Fuller and Isamu Noguchi: Best of Friends</td>
<td>Shoji Sadao</td>
<td>60</td>
</tr>
<tr>
<td>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</td>
<td>Louise Allison Cort Bert Winther-Tamaki Bruce J. Altshuler Ryu Nimi</td>
<td>29.95</td>
</tr>
</tbody>
</table>

This glitch occurs because the XML is comparing the price and the discount as strings: That is the default behavior for the comparison operators when the operands are untyped. The string 5.95 is greater than the string 29.95, so the comparison is returning false instead of true. Adding types to my schema, as in Listing 8, corrects this problem.
Listing 8. Scenario 2 XSLT, final version

```xml
<xsl:function name="dtyf:calculate-price" as="xs:decimal">
    <xsl:param name="price" as="xs:decimal"/>
    <xsl:param name="discount" as="xs:decimal?"/>
    <xsl:choose>
        <xsl:when test="$discount > 0 and $discount &lt; $price"> 
            <xsl:sequence select="$price - $discount"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:sequence select="$price"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:function>
```

I specify that both parameters are decimal numbers, and I use the occurrence indicator `?` to allow an empty sequence for the discount, because it might be missing. In the revised XSLT, the values of the price and discount elements are converted to decimal numbers when the function is called. (This happens automatically when the input data is untyped—that is, does not have a schema.) Now, the XML is comparing price and discount as numbers, and I get the correct results, shown in Figure 6.

Figure 6. Results of final Scenario 2 XSLT on morebooks.xml

<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isamu Noguchi Sculptural Design</td>
<td>Vitra Design Museum</td>
<td>125</td>
</tr>
<tr>
<td>The East-West House: Noguchi's Childhood in Japan</td>
<td>Christy Hale</td>
<td>8</td>
</tr>
<tr>
<td>Buckminster Fuller and Isamu Noguchi: Best of Friends</td>
<td>Shoji Sadao</td>
<td>60</td>
</tr>
</tbody>
</table>

My rounding error is gone, too, because by default, the subtraction operator was treating numbers like `<xs:double>` values, which are floating-point numbers that are not treated with as much precision as `<xs:decimal>` values.

Using schemas to improve your stylesheets

The previous examples use XML schema built-in types, but they don’t require a schema for the input document. Now, look at how using XML schemas can make the XSLT even more robust. Schema awareness is an optional feature of XSLT processors, and the stylesheets in this section must be run with a schema-aware XSLT processor. The examples in this article were tested using Saxon-EE (see Related topics for a link to more information).

Importing and using schemas

Suppose you have a schema for the books.xml document, named `books.xsd`, shown in Listing 9. It defines all the data types and cardinalities of the elements. A complete explanation of XML Schema is outside the scope of this article: I recommend checking out the XML Primer (see Related topics) for a basic introduction.
Listing 9. Books schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.datypic.com/books/ns"
  xmlns="http://www.datypic.com/books/ns" elementFormDefault="qualified">
  <xs:element name="books" type="BooksType"/>
  <xs:complexType name="BooksType">
    <xs:sequence>
      <xs:element ref="book" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="book" type="BookType"/>
  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="author" maxOccurs="unbounded"/>
      <xs:element ref="price"/>
      <xs:element ref="discount" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="author" type="AuthorType"/>
  <xs:element name="price" type="xs:decimal"/>
  <xs:element name="discount" type="xs:decimal"/>
  <xs:complexType name="AuthorType">
    <xs:choice>
      <xs:element ref="person"/>
      <xs:element ref="institution"/>
    </xs:choice>
  </xs:complexType>
  <xs:element name="person" type="PersonType"/>
  <xs:element name="institution" type="xs:string"/>
  <xs:complexType name="PersonType">
    <xs:sequence>
      <xs:element ref="first"/>
      <xs:element ref="middle" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="last"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

When you use schemas with your stylesheets, you can use the additional sequence types listed in Table 4.

**Table 4. Schema-aware sequence types**

<table>
<thead>
<tr>
<th>Sequence type example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>element(*,BookType)</td>
<td>Any element whose type is BookType</td>
</tr>
<tr>
<td>element(book,BookType)</td>
<td>Any element named book whose type is BookType</td>
</tr>
<tr>
<td>schema-element(book)</td>
<td>Any element validated against a global declaration for book in the schema or a member of its substitution group</td>
</tr>
<tr>
<td>attribute(*,ISBNType)</td>
<td>Any attribute whose type is ISBNType</td>
</tr>
</tbody>
</table>
Scenario 3: Detecting invalid paths

One extremely useful feature of using schemas is that it tells you when you have used an invalid name or path in your XPath expressions. Listing 10 shows an XSLT that contains several subtle errors.

Listing 10. Scenario 3 XSLT, original version

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xpath-default-namespace="http://www.datypic.com/books/ns">
    <xsl:output method="html"/>
    <xsl:template match="books">
        <html>
            <body>
                <table border="1">
                    <xsl:apply-templates/>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="book">
        <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="author/last"/></td>
            <td><xsl:value-of select="author/first"/></td>
            <td><xsl:value-of select="prce"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

My results, in Figure 7, are missing both author names and prices. Without a schema, I have to figure out by trial and error what the problems are.

Figure 7. Results of original Scenario 3 XSLT on books.xml

Making my stylesheet schema aware will help. First, I have to import the schema, as in Listing 11, giving it the file name (absolute or relative) of the schema and the target namespace of that schema. In addition, to get the error checking I want, I have to change the sequence types used in my match attributes to use schema-element().

Listing 11. Scenario 3 XSLT, revised version

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xpath-default-namespace="http://www.datypic.com/books/ns">
```

<table>
<thead>
<tr>
<th>attribute(isbn,ISBNType)</th>
<th>Any attribute named isbn whose type is ISBNType</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema-attribute(isbn)</td>
<td>Any attribute validated against a global declaration for isbn in the schema or a member of its substitution group</td>
</tr>
</tbody>
</table>
Now I get three error messages:

- The complex type AuthorType does not allow a child element named last
- The complex type AuthorType does not allow a child element named first
- The complex type BookType does not allow a child element named prce

The first two error messages tell me that I got the path wrong; I forgot about the person element that is a child of author. The third error message makes me realize that I misspelled price. Listing 12, amended to use the correct paths, gives the complete output.

**Listing 12. Scenario 3 XSLT, final version**

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ns="http://www.datypic.com/books/ns"
  xslt-namespace="http://www.datypic.com/books/ns">
  <xsl:output method="html"/>
  <xsl:import-schema namespace="http://www.datypic.com/books/ns"
    schema-location="books.xsd"/>
  <xsl:template match="books">
    <table border="1">
      <xsl:apply-templates/>
    </table>
  </xsl:template>
  <xsl:template match="schema-element(book)">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author/person/last"/></td>
      <td><xsl:value-of select="author/person/first"/></td>
      <td><xsl:value-of select="price"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```
Scenario 4: Validating output

In some cases, you might want to ensure that you are generating valid output. This is especially true in the case of structured data conversion but can also apply to a case where you are generating XHTML. To output XHTML, I might decide to just take the code from Listing 12 and change the output method to xhtml. In this case, the result will look right in the browser. But maybe it needs to be valid XHTML, either because it’s going to be input to some other stylesheet or process that requires valid input or because I am a stickler for following standards, which I am!

To check the validity of output, I first have to import the schema, just as I did for the input document schema. I also tell the processor that I want to validate the output by adding the xsl:validation="strict" attribute to the root element. Doing so causes all of the contents of html to be validated.

Another change I incorporate is to make the XHTML namespace the default namespace because to be valid, the output must be in the correct namespace. Listing 13 shows these changes.

Listing 13. Scenario 4 XSLT, revised from Scenario 3

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml"
    xpath-default-namespace="http://www.datypic.com/books/ns">

    <xsl:output method="xhtml"/>
    <xsl:import-schema namespace="http://www.datypic.com/books/ns"
        schema-location="books.xsd"/>
    <xsl:import-schema namespace="http://www.w3.org/1999/xhtml"
        schema-location="xhtml1-strict.xsd"/>

    <xsl:template match="books">
        <html xsl:validation="strict">
            <body>
                <table border="1">
                    <xsl:apply-templates/>
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="schema-element(book)">
        <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="author/person/last"/></td>
            <td><xsl:value-of select="author/person/first"/></td>
            <td><xsl:value-of select="price"/></td>
        </tr>
    </xsl:template>
</xsl:stylesheet>
```

When I run this, I get the following error message:

- In content of element <html>: The content model does not allow element <body> to appear here. Expected: {http://www.w3.org/1999/xhtml}head
The error message is telling me that I am missing a head element. After several iterations, I end up with the final XSLT in Listing 14, which generates strictly valid XHTML.

### Listing 14. Scenario 4 XSLT, final version

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
xpath-default-namespace="http://www.datypic.com/books/ns">
  <xsl:output method="xhtml"/>
  <xsl:import-schema namespace="http://www.datypic.com/books/ns"
schema-location="books.xsd"/>
  <xsl:import-schema namespace="http://www.w3.org/1999/xhtml"
schema-location="xhtml1-strict.xsd"/>

  <xsl:template match="books">
    <html xsl:validation="strict">
      <head><title>Books</title></head>
      <body>
        <table border="1">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="schema-element(book)">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author/person/last"/></td>
      <td><xsl:value-of select="author/person/first"/></td>
      <td><xsl:value-of select="price"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

### Conclusion

As you saw from the examples in this article, XSLT stylesheets can sometimes silently fail or provide incorrect results. In these simple examples, you're likely to recognize the problems and be able to fix them easily. Input documents and XSLT stylesheets, however, are usually much more complex. Without explicit types and imported schemas, XSLTs can be a challenge to debug.

In addition, test data documents don't always contain all the permutations that you can encounter in real input data. Explicitly specifying schema types can bring out problems with the data that you didn't even encounter during testing. Specifying the types of parameters and return values also results in better documented functions and templates, making your code easier to maintain and easier for others to understand.

If you are still unconvinced, I recommend adding types and schemas to some of your existing XSLT stylesheets. You might be surprised what you find!
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample XSLT stylesheets for this article</td>
<td>examples.zip</td>
<td>10KB</td>
</tr>
</tbody>
</table>
Related topics

- **Saxon**: Download the schema-aware XSLT 2.0 processor used to test the examples in this article.
- **XML area on developerWorks**: Find the resources you need to advance your skills in the XML arena, including DTDs, schemas, and XSLT. See the XML technical library for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- **IBM certification**: Find out how you can become an IBM-Certified Developer.
- **developerWorks on Twitter**: Join today to follow developerWorks tweets.
- **IBM product evaluation versions**: Get your hands on application development tools and middleware products.

© Copyright IBM Corporation 2012

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)