Using types and schemas to improve your XSLT 2.0 stylesheets

Priscilla Walmsley <pwalmsley@datypic.com>

May 15, 2012

Abstract

Using the type-aware and schema-aware features of XSLT 2.0 can greatly assist the debugging of a stylesheet, thereby improving its quality and its robustness in handling all input data. This article explains 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. It provides examples of XSLT stylesheets that contain errors that would not be caught if schema-aware features were not in use, and shows how explicitly specifying types results in useful error messages.

Table of Contents

Introduction

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

XSLT 2.0 is a big improvement over 1.0, and strong typing and schema-awareness are very useful added features. Declaring types for values in an XSLT stylesheet allows the processor to tell you when you are making incorrect assumptions about the data type of the XML content, or the number of occurrences of a particular element or attribute. This results in much more useful error messages. Importing XML Schemas takes it a step further, giving the processor enough information about the input document structure to inform you when you have invalid XPaths, rather than simply returning nothing. It also provides information about data types in the content, preventing you from performing operations that do not 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 this, you prefix the type name with, for example, `xs:` and declare the `xs:` namespace at the top of your stylesheet. The most commonly used XML Schema data types are listed in Table 1.

Table 1. Common XML Schema data types

Data type name	Description	Example(s)
`string`	Any text string	abc, this is a string
`integer`	An integer of any size	1, 2
`decimal`	A decimal number	1.2, 5.0
`double`	A double-precision floating point number	1.2, 5.0
`date`	A date, in `YYYY-MM-DD` format	2009-12-25
`time`	A time, in `HH:MM:SS` format	12:05:04
`boolean`	A true/false value	true, false
`anyAtomicType`	A value of any of the simple types	a string, 123, false, 2009-12-25

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 there is no schema associated with the input document).

Sequence types representing XML nodes

You can also use 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

Sequence type	Description
`element()`	Any element
`element(book)`	Any element named `book`
`attribute()`	Any attribute
`attribute(isbn)`	Any attribute named `isbn`
`text()`	Any text node
`node()`	A node of any kind (element, attribute, text node, etc.)
`item()`	Either a node of any kind or an atomic value of any kind (e.g. a string, integer, etc.)

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

```
<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 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

Occurrence indicator	Description
`*`	Zero to many
`?`	Zero to one
`+`	One to many
(no occurrence indicator)	One and only one

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

```
<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 may be wondering how the use of `as` attributes improves your stylesheets. Let's look at a couple of examples. All of the examples in this article can be downloaded as a zip file.

Scenario 1: Using types to enforce cardinalities

Example 1 shows an input document that contains two books.

Example 1. Sample book input, books.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>
```

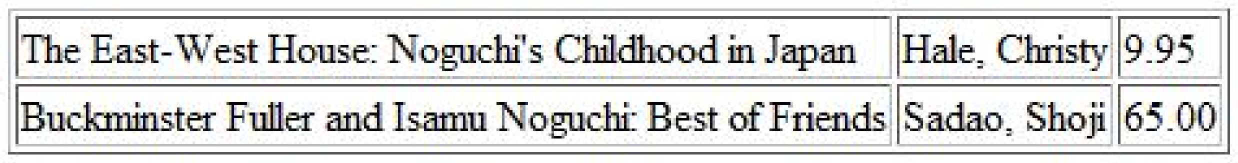
Example 2 is a stylesheet that will transform book input into HTML.

Example 2. Scenario 1 XSLT, original version

```
<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



The East-West House: Noguchi's Childhood in Japan	Hale, Christy	9.95
Buckminster Fuller and Isamu Noguchi: Best of Friends	Sadao, Shoji	65.00

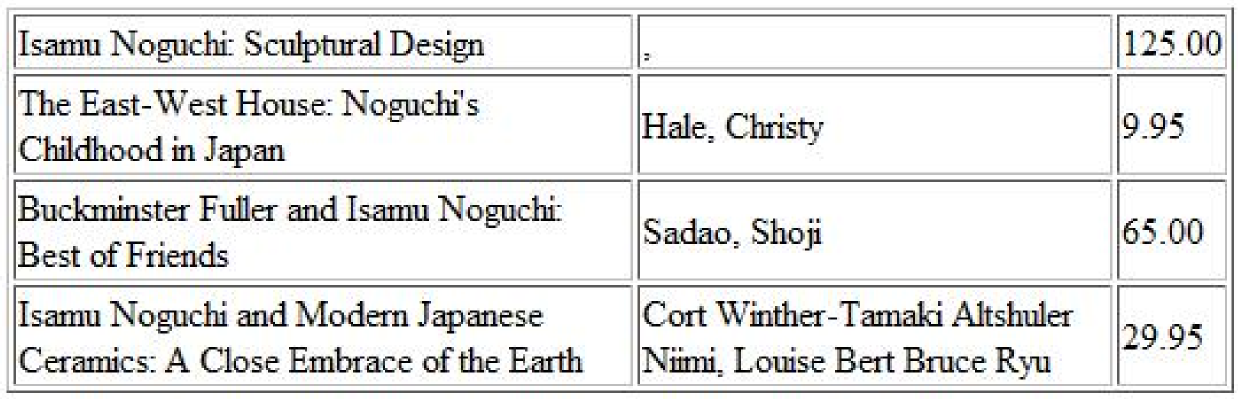
However, suppose I try to run it on more varied test data, `morebooks.xml` (Example 3).

Example 3. More varied input document, morebooks.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>
   <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>
 <book>
  <title>Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth</title>
  <author>
   <person>
    <first>Louise</first>
    <middle>Allison</middle>
    <last>Cort</last>
   </person>
  </author>
  <author>
   <person>
    <first>Bert</first>
    <last>Winther-Tamaki</last>
   </person>
  </author>
  <author>
   <person>
    <first>Bruce</first>
    <middle>J.</middle>
    <last>Altshuler</last>
   </person>
  </author>
  <author>
   <person>
    <first>Ryu</first>
    <last>Niimi</last>
   </person>
  </author>
  <price>29.95</price>
  <discount>5.95</discount>
 </book>
</books>
```

Figure 2 shows that it has incomplete results. For books that have an institution as an author, it is missing. For ones 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



Isamu Noguchi: Sculptural Design	,	125.00
The East-West House: Noguchi's Childhood in Japan	Hale, Christy	9.95
Buckminster Fuller and Isamu Noguchi: Best of Friends	Sadao, Shoji	65.00
Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth	Cort Winther-Tamaki Altshuler Niimi, Louise Bert Bruce Ryu	29.95

To create a more robust stylesheet, I can amend the function as shown in Example 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.

Example 4. Scenario 1 XSLT, revised function

```
<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), and
- `A sequence of more than one item is not allowed as the first argument of dtyf:format-person-name()` (for multiple authors)

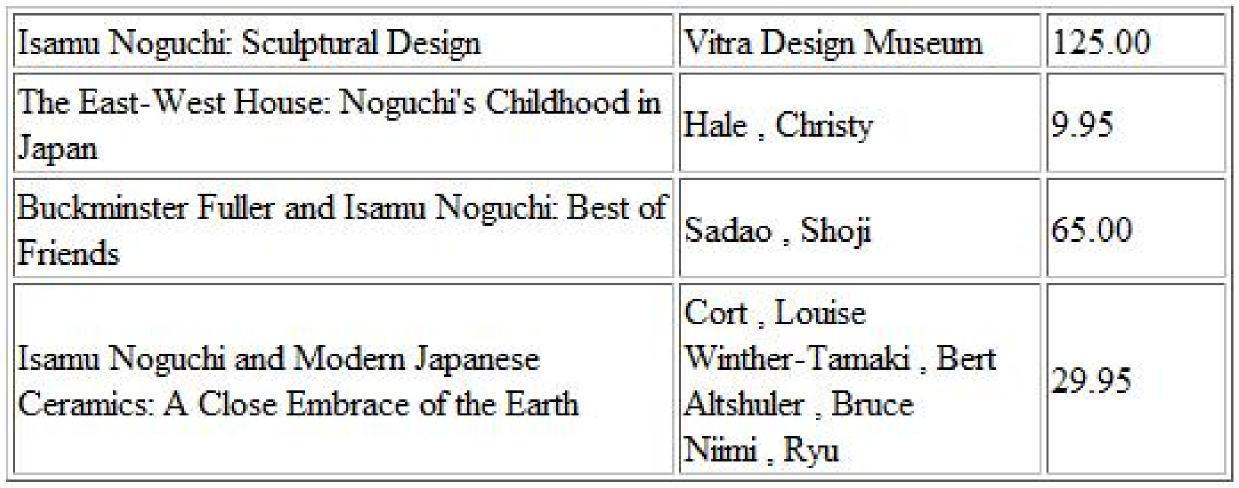
Based on these error messages I know I need to change the XSLT as shown in Example 5 to handle both institutional authors and more than one author.

Example 5. Scenario 1 XSLT, final version

```
<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:for-each select="author">
    <xsl:choose>
     <xsl:when test="person">
      <xsl:value-of select="dtyf:format-person-name(person)"/>
     </xsl:when>
     <xsl:when test="institution">
      <xsl:value-of select="institution"/>
     </xsl:when>
    </xsl:choose>
    <xsl:if test="position() != last()">
     <br/>
    </xsl:if>
   </xsl:for-each>
  </td>
 <td><xsl:value-of select="price"/></td>
 </tr>
</xsl:template>

<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>
</xsl:stylesheet>
```

My results, shown in Figure 3, look a lot better.

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



Isamu Noguchi: Sculptural Design	Vitra Design Museum	125.00
The East-West House: Noguchi's Childhood in Japan	Hale , Christy	9.95
Buckminster Fuller and Isamu Noguchi: Best of Friends	Sadao , Shoji	65.00
Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth	Cort , Louise Winther-Tamaki , Bert Altshuler , Bruce Niimi , Ryu	29.95

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

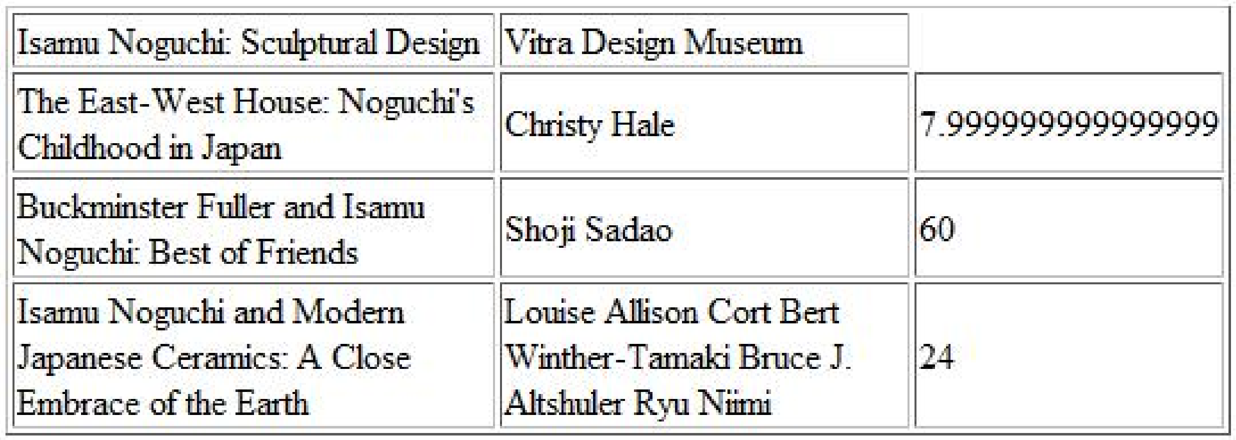
Continuing with our book example, Example 6 shows an XSLT that now takes into account the `discount` element when calculating the price.

Example 6. Scenario 2 XSLT, original version

```
<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 shown in Figure 4. This is because the value of `$discount` is an empty sequence, and any arithmetic operation on the empty sequence also returns the empty sequence.

Figure 4. Results of original Scenario 2 XSLT on morebooks.xml



Isamu Noguchi: Sculptural Design	Vitra Design Museum	
The East-West House: Noguchi's Childhood in Japan	Christy Hale	7.999999999999999
Buckminster Fuller and Isamu Noguchi: Best of Friends	Shoji Sadao	60
Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth	Louise Allison Cort Bert Winther-Tamaki Bruce J. Altshuler Ryu Niimi	24

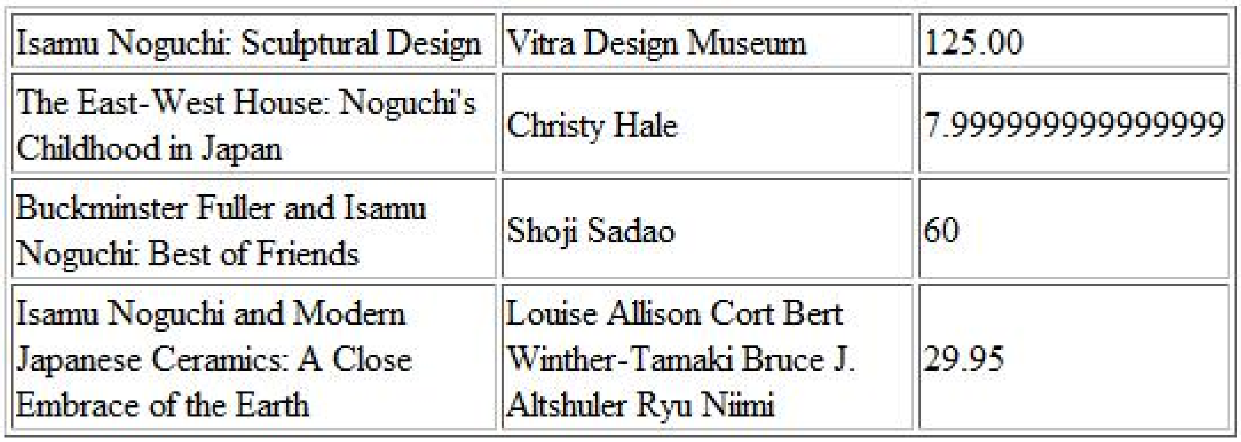
I consider just comparing $discount to 0 to fix this, but then I decide that the function will be even more robust if I make sure the discount is less than the price. I don't want any negative prices in my table. So I also include the comparison `$discount < $price`, as shown in Example 7.

Example 7. Scenario 2 XSLT, revised version

```
<xsl:function name="dtyf:calculate-price">
<xsl:param name="price"/>
<xsl:param name="discount"/>
<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>
```

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



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

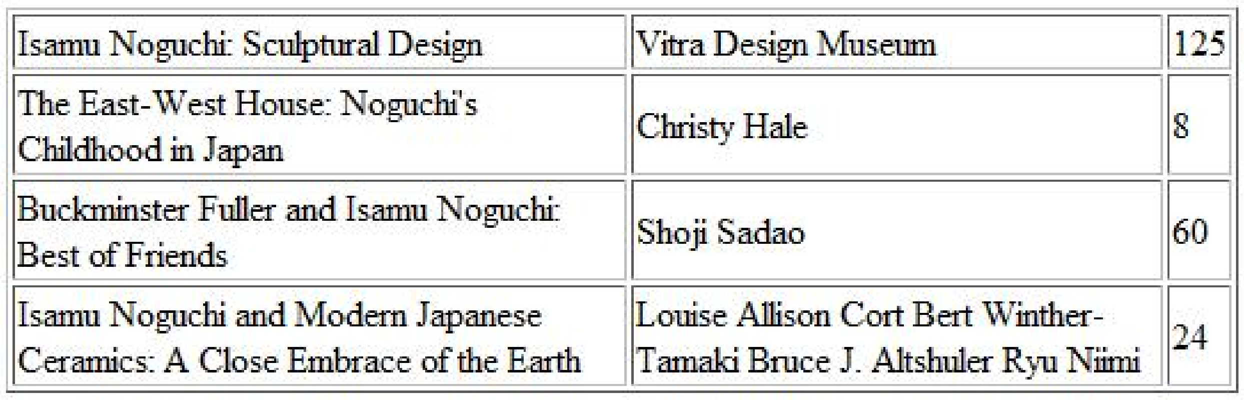
This is because it is comparing the price and the discount as strings; that is the default behavior for the comparison operators when the operands are untyped. The string `5.95` is greater than the string `29.95`, so the comparison is returning false instead of true. Adding types to my schema as shown in Example 8 will correct this problem.

Example 8. Scenario 2 XSLT, final version

```
<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, since it may 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, i.e. does not have a schema.) Now it 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



Isamu Noguchi: Sculptural Design	Vitra Design Museum	125
The East-West House: Noguchi's Childhood in Japan	Christy Hale	8
Buckminster Fuller and Isamu Noguchi: Best of Friends	Shoji Sadao	60
Isamu Noguchi and Modern Japanese Ceramics: A Close Embrace of the Earth	Louise Allison Cort Bert Winther-Tamaki Bruce J. Altshuler Ryu Niimi	24

My rounding error is gone too. That's 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, let's take a 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.

Importing and using schemas

Suppose we have a schema for the books.xml document, named books.xsd, shown in Example 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 Schema Primer for a basic introduction.

Example 9. Books schema

```
<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:element name="first" type="xs:string"/>
 <xs:element name="middle" type="xs:string"/>
 <xs:element name="last" type="xs:string"/>

</xs:schema>
```

When you are using schemas with your stylesheets, you can use additional kinds of sequence types listed in Table 4.

Table 4. Schema-aware sequence types

Sequence type example	Meaning
`element(*, BookType)`	any element whose type is `BookType`
`element(book, BookType)`	any element named book and whose type is `BookType`
`schema-element(book)`	any element validated against a global declaration for `book` in the schema, or a member of its substitution group
`attribute(*, ISBNType)`	any attribute whose type is `ISBNType`
`attribute(isbn, ISBNType)`	any attribute named `isbn` and whose type is `ISBNType`
`schema-attribute(isbn)`	any attribute validated against a global declaration for `isbn` in the schema

Scenario 3: Detecting invalid paths

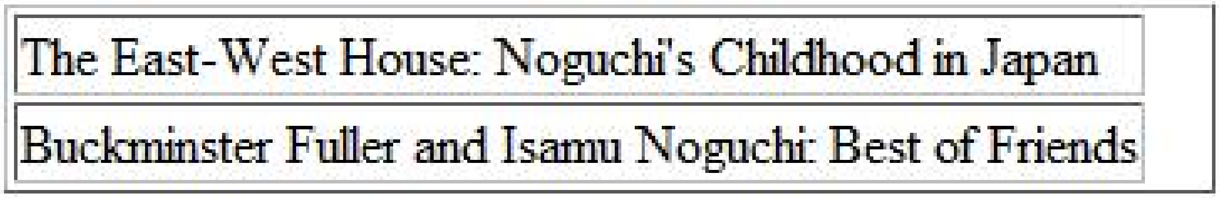
One extremely useful feature of using schemas is that it tells you when you have used an invalid name or path in your XPath expressions. Example 10 shows an XSLT that contains several subtle errors.

Example 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, shown 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



The East-West House: Noguchi's Childhood in Japan
Buckminster Fuller and Isamu Noguchi: Best of Friends

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

Example 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">

 <xsl:output method="html"/>
 <xsl:import-schema namespace="http://www.datypic.com/books/ns"
                    schema-location="books.xsd"/>
 <xsl:template match="schema-element(books)">
  <html>
   <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/last"/></td>
  <td><xsl:value-of select="author/first"/></td>
  <td><xsl:value-of select="prce"/></td>
 </tr>
</xsl:template>

</xsl:stylesheet>
```

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 tells 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". Example 12, amended to use the correct paths, gives the complete output.

Example 12. Scenario 3 XSLT, final 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:import-schema namespace="http://www.datypic.com/books/ns"
                   schema-location="books.xsd"/>
<xsl:template match="books">
 <html>
  <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>
```

Scenario 4: Validating output

In some cases, you may 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 Example 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 is 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 like 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. This will cause all of the contents of `html` to be validated.

Another change I make is to make the XHTML namespace the default namespace, since in order to be valid the output must be in the correct namespace. These changes are shown in Example 13.

Example 13. Scenario 4 XSLT, revised from Scenario 3

```
<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 shown in Example 14, which generates strictly valid XHTML.

Example 14. Scenario 4 XSLT, final version

```
<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>
```

Conclusions

As we have seen from the examples in this article, XSLT stylesheets can sometimes silently fail or provide incorrect results. In these simple examples, I am likely to recognize the problems and be able to fix them easily. However, input documents and XSLT stylesheets 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 can be encountered in real input data. When schema types are explicitly specified, it can bring out problems with the data that were not even encountered during testing.

Specifying the types of parameters and return values also results in better documented functions and templates. This makes 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 may be surprised what you find!

About the author

Priscilla Walmsley serves as Managing Director and Senior Developer at Datypic. She specializes in XML technologies, architecture and implementation. She is the author of *Definitive XML Schema* (Prentice Hall, 2012), and *XQuery* (O'Reilly Media, 2015). In addition, she co-authored *Web Service Contract Design and Versioning for SOA* (Prentice Hall, 2008).

About Datypic

Datypic provides development services and training, specializing in XML, content management and electronic publishing. We are experts in XML-related technologies such as XML Schema, XSLT and XQuery, and have extensive experience with software development and implementation.

We participate in projects ranging from one day to many months, anywhere in the world. We can arrange to work remotely or at your site, whichever you prefer.

For more information, please read about our services at datypic.com.

About this article

A prior version of this article was first published by IBM developerWorks. Its current version number is 2.0 and it was last updated on April 30, 2014.