

## SYBEX Appendix

# Mastering™ XSLT

Chuck White

## Appendix E: XSLT Functions

Copyright © 2002 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

ISBN: 0-7821-4094-7

SYBEX and the SYBEX logo are either registered trademarks or trademarks of SYBEX Inc. in the USA and other countries.

TRADEMARKS: Sybex has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer. Copyrights and trademarks of all products and services listed or described herein are property of their respective owners and companies. All rules and laws pertaining to said copyrights and trademarks are inferred.

This document may contain images, text, trademarks, logos, and/or other material owned by third parties. All rights reserved. Such material may not be copied, distributed, transmitted, or stored without the express, prior, written consent of the owner.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturers. The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Sybex Inc.  
1151 Marina Village Parkway  
Alameda, CA 94501  
U.S.A.  
Phone: 510-523-8233  
[www.sybex.com](http://www.sybex.com)



## Appendix E

# XSLT Functions

THIS APPENDIX PROVIDES A reference to XSLT functions. XSLT functions are different from XPath functions: An XSLT processor can use both, but an XPath processor cannot necessarily use XSLT functions. Therefore, we are presenting these functions separately from Appendix A, “XPath,” which deals entirely with XPath.

All but one of the examples in this reference use either Listing E.1 or Listing E.2 as their source documents. The only difference between the two is that Listing E.2 includes a DTD so that we can demonstrate the procedures involved with accessing and manipulating an entity URI, the last function shown in the appendix.

*TIP* The most authoritative reference on the Internet for XSLT functions is obviously the XSLT specification at the W3C site, since it contains the actual specification and is therefore the final arbiter of disputes. It can be found at [www.w3.org/TR/xslt](http://www.w3.org/TR/xslt).

### LISTING E.1: SOURCE DOCUMENT WITHOUT DTD

```
<?xml version="1.0"?>
<orders>
  <order ordernum="0001">
    <customer>
      <firstname>John</firstname>
      <lastname>Capitalist</lastname>
      <phone>(650) 555-1212</phone>
    </customer>
  </order>
  <order ordernum="0002">
    <customer>
      <firstname>Jane</firstname>
      <lastname>Socialist</lastname>
      <phone>(415) 555-1212</phone>
    </customer>
  </order>
```

```

<order ordernum="0003">
  <customer>
    <firstname>Joe</firstname>
    <lastname>Reformer</lastname>
    <phone>(212) 555-1212</phone>
  </customer>
</order>
<order ordernum="0004">
  <customer>
    <firstname>John</firstname>
    <lastname>Conformer</lastname>
    <phone>(312) 555-1212</phone>
  </customer>
</order>
</orders>

```

---

### LISTING E.2: SOURCE DOCUMENT WITH DTD

```

<?xml version="1.0"?>
<!DOCTYPE advertisement SYSTEM "ad.dtd">
<advertisement>
  <id>Retail</id>
  <rundate id="ad_001">09/09/99</rundate>
  <text>only text</text>
  <image src="tumeric"/>
</advertisement>

```

---

**NOTE** Listing E.1 and Listing E.2 are downloadable from [www.sybex.com](http://www.sybex.com) as LC01.xml and LC02.xml, respectively.

### **current()**

The `current()` function provides a way to conveniently get at the current node by returning its value. This is helpful for those instances when you don't want the context node as your "base of operations." The example below generates a unique id for the current node.

### **EXAMPLE**

The source document is Listing E.1.

```

<xsl:for-each select="order">
  <tr>
    <td>
      <xsl:apply-templates
        select="@ordernum[ $N ]"/>
    </td>
  </tr>
</xsl:for-each>

```

```

</td>
<td>
  <xsl:value-of select="generate-id(current())"/>
</td>
<td>
  <xsl:apply-templates
    select="customer/firstname[ $N ]"/>
</td>
<td>
  <xsl:apply-templates
    select="customer/firstname[ $N ]"/>
  <xsl:apply-templates
    select="customer/lastname[ $N ]"/>
</td>
<td>
  <xsl:apply-templates
    select="customer/phone[ $N ]"/>
</td>
</tr>
</xsl:for-each>

```

#### RESULT TREE FRAGMENT

```

<tr>
<td>0001</td>
<td>IDAZZKIB</td>
<td>JohnCapitalist</td>
<td>(650) 555-1212</td>
</tr>

```

#### ***document(uri, base-uri)***

The `document()` function is used to import an external XML document. This tool is useful for applying a key tenet of XML document design, modularization, which simply means the development of multiple documents that could conceivably be combined into one document. If you are at all familiar with the XHTML specifications, you have undoubtedly seen modularization at work, since the XHTML DTDs make great use of modular design principles. You can do the same thing with XSLT, as shown in Chapter 11, “Managing Multiple Documents and Modularization.” The function can take two parameters in its arguments, a URI and an optional base URI.

#### EXAMPLE

The source document is Listing E.1.

```
<xsl:variable name="showDoc" select="document(LC02.xml)"/>
```

After setting up a variable that brings in the document, you can perform a node-test on the document, as follows:

```
<xsl:copy-of select="$showDoc/advertisement/id"/>
```

The preceding line of code simply accesses a node in a routine manner, with the one difference being that the root node is now the root node of the imported document, in the form of a variable, rather than the root node of the original source document. For a thorough discussion of this function, read Chapter 11. The result looks like this:

```
<id>Retail</id>
```

### ***element-available()***

The `element-available()` function is designed to help deal with versioning issues involving XSLT, and is a good way to test if a proprietary element (or extension element) is available. It is used to test whether a specific XSLT element is available and whether that element is an actual instruction that can be used in the stylesheet. If it isn't, you can write a simple routine for avoiding errors to deal with it. One thing that is unique about this particular function is that it doesn't test the source document for anything or work with node-sets. Rather, it queries the XSLT processor itself about its capabilities or about the implementation of XSLT you are using. If you want your code to be portable, you should use this function whenever you are using extension elements, and offer alternative processing instructions if an element isn't available.

The following are XSLT instructions that you can test for in XSLT 1.0:

<code>xsl:apply-imports</code>	<code>xsl:fallback</code>
<code>xsl:apply-templates</code>	<code>xsl:for-each</code>
<code>xsl:attribute</code>	<code>xsl:if</code>
<code>xsl:call-template</code>	<code>xsl:message</code>
<code>xsl:choose</code>	<code>xsl:number</code>
<code>xsl:comment</code>	<code>xsl:processing-instruction</code>
<code>xsl:copy</code>	<code>xsl:text</code>
<code>xsl:copy-of</code>	<code>xsl:value-of</code>
<code>xsl:element</code>	<code>xsl:variable</code>

### **EXAMPLE**

The source document is Listing E.1.

```
<td>
  <xsl:if test="element-available('xsl:text')">
    <xsl:text>The element is available</xsl:text>
  </xsl:if>
</td>
```

### **RESULT TREE FRAGMENT**

```
<td>The element is available</td>
```

## ***format-number()***

The `format-number()` function does what it says: it formats a number according to criteria spelled out in the argument. The format is indicated through a pattern identified in Tables E.1, E.2, and E.3, which follow the JDK 1.1 syntax (JDK is Java Developer's Kit) for pattern-based number formatting.

**TABLE E.1: PATTERNS FOR NUMBER FORMATTING**

<b>PATTERN TYPE</b>	<b>PATTERN</b>
Subpattern	{;subpattern}
Subpattern	{prefix}integer{.fraction}{suffix}
Prefix	'\u0000'..' \uFFFF' - specialCharacters
Suffix	'\u0000'..' \uFFFF' - specialCharacters
Integer	'#'* '0'* '0'
Fraction	'0'* '#'*

**TABLE E.2: PATTERN NOTATION FOR NUMBER FORMATTING**

<b>CHARACTER</b>	<b>MEANING</b>
X*	Zero or more instances of X
(X   Y)	Either X or Y
X..Y	Any character from X up to Y, inclusive
S - T	Characters in S, except those in T

**TABLE E.3: SPECIAL CHARACTERS USED IN SUBPATTERNS FOR NUMBER FORMATTING**

<b>SYMBOL</b>	<b>MEANING</b>
0	A digit.
#	A digit; zero shows as absent.
.	Placeholder for decimal separator.
,	Placeholder for grouping separator.
;	Separates formats.
-	Default negative prefix.
%	Multiply by 100 and show as percentage.

*Continued on next page*

**TABLE E.3: SPECIAL CHARACTERS USED IN SUBPATTERNS FOR NUMBER FORMATTING** (*continued*)

SYMBOL	MEANING
?	Multiply by 1000 and show as per mille.
¤	Currency sign; replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
X	Any other characters can be used in the prefix or suffix.
'	Used to quote special characters in a prefix or suffix.

In the following example, note that the source document contains an `ordernumber` attribute that has leading zeros. So we create a variable for it, use it for the `format-number`'s first parameter, and eliminate the leading zeros by using the `#` symbol, which suppresses zeros.

**NOTE** For the Java documentation on number formatting, visit [java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html](http://java.sun.com/products/jdk/1.1/docs/api/java.text.DecimalFormat.html).

**NOTE** It doesn't matter whether you are working on a Java-based XSLT processor or not—the number formatting follows the patterns described in JDK 1.1.

#### EXAMPLE

The source document is Listing E.1.

```
<xsl:for-each select="order">
  <tr>
    <td>
      <xsl:variable name="formattedNumber"
        select="@ordernum[position()=$N ]"/>
      <xsl:value-of
        select="format-number($formattedNumber,
          '###0')"/></td>
    <td>
      <xsl:value-of
        select="generate-id(current())"/>
    </td>
    <td>
      <xsl:apply-templates
        select="customer/firstname[ $N ]"/>
      <xsl:apply-templates
        select="customer/lastname[ $N ]"/>
    </td> <td>
      <xsl:apply-templates
        select="customer/phone[ $N ]"/>
    </td>
  </tr>
</xsl:for-each>
```

```

    </td>
  </tr>
</xsl:for-each>

```

### RESULT TREE FRAGMENT

```

<tr>
  <td>1</td>
  <td>IDA4GKIB</td>
  <td>JohnCapitalist</td>
  <td>(650) 555-1212</td>
</tr>
<tr>
  <td>2</td>
  <td>IDAEHKIB</td>
  <td>JaneSocialist</td>
  <td>(415) 555-1212</td>
</tr>
<tr>
  <td>3</td>
  <td>IDAKHKIB</td>
  <td>JoeReformer</td>
  <td>(212) 555-1212</td>
</tr>
<tr>
  <td>4</td>
  <td>IDAQHKIB</td>
  <td>JohnConformer</td>
  <td>(312) 555-1212</td>
</tr>

```

### *function-available()*

The `function-available()` function is designed to help deal with versioning issues involving XSLT. It is used to test whether a specific XSLT or XPath function is available to the processor. If it isn't, you can write a routine for avoiding errors to deal with it. One thing that is unique about this particular function is that it doesn't test the source document for anything or work with node-sets. Rather, it queries the XSLT processor itself about its capabilities or about the implementation of XSLT you are using.

### EXAMPLE

The source document is Listing E.1.

```

<xsl:if test="function-available('name')">
  <xsl:text>The name function is available</xsl:text>
</xsl:if>

```

### RESULT TREE FRAGMENT

```

<td>The function is available</td>

```

***generate-id()***

The `generate-id()` function generates unique ids for your result set nodes to make it more convenient to add unique ids to manipulate objects. Having an `id` attribute with unique ids is useful when manipulating an XML document using the DOM, a common occurrence in dynamic SVG and dynamic HTML. You might also want to use it for databases, although this function generates letters, which can be a problem if your database requires integers for its ids.

**EXAMPLE**

The source document is Listing E.1.

```
<xsl:template match="orders">
  <xsl:for-each select="order">
    Order Number:<xsl:value-of
      select="@ordernum"/>
    <xsl:text>; ID Number:</xsl:text><xsl:value-of
      select="generate-id(current())"/>
    </xsl:for-each>
  </xsl:template>
```

**RESULT TREE FRAGMENT**

```
Order Number:0001; ID Number:IDA2MF4
Order Number:0002; ID Number:IDACNF4
Order Number:0003; ID Number:IDAINF4
Order Number:0004; ID Number:IDAONF4
```

***key(name, value)***

The `key()` function searches for named keys created using the `xsl:key` element. The first parameter is the name of a key defined using `xsl:key`. The second parameter is an expression. Its value in general will probably be a variable, but our example shows an explicit expression.

**EXAMPLE**

The source document is Listing E.1. Given the following `xsl:key` statement at the top level of the stylesheet,

```
<xsl:key name="orderKey" match="order" use="@ordernum"/>
```

you can pass the function the key information like so:

```
<xsl:template match="orders">
  <td>
    <xsl:if test="key('orderKey', '0001')">
      <xsl:value-of select="order/@ordernum"/>
    </xsl:if>
  </td>
</xsl:template>
```

**RESULT TREE FRAGMENT**

```
<td>0001</td>
```

***system-property()***

The `system-property()` function provides information about the XSLT processor you are working with.

**EXAMPLE**

The source document is Listing E.1.

```
<h1>Example using an XSLT processor made by <xsl:value-of select=
  ↪ "system-property('xsl:vendor')"/></h1>
```

**RESULT TREE FRAGMENT**

```
<h1>Example using an XSLT processor made by Apache Software Foundation</h1>
```

***unparsed-entity-uri()***

The `unparsed-entity-uri()` function provides information about unparsed entities in the source document's DTD. The string argument contains the name of entity you seek, and the string must resolve to an XML name. The example that follows uses the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT advertisement (id | rundate | image | text)*>
<!ELEMENT id (#PCDATA)>
<!ELEMENT rundate (#PCDATA)>
<!ATTLIST rundate
  id ID #REQUIRED
>
<!ELEMENT text (#PCDATA)>
<!ENTITY tumeric SYSTEM "logo.png" NDATA PNG>
<!NOTATION PNG SYSTEM "iexplore.exe">
<!ELEMENT image (#PCDATA)>
<!ATTLIST image
  src ENTITY #IMPLIED
  alt CDATA #IMPLIED
>
```

**WARNING** *Unparsed entities are used only within attribute values, and their syntax is not what you might expect. You don't, for example, refer to an entity named `entity` as `$entity`; in your attribute value. In other words, when using an unparsed entity as an attribute value in your source document as we do in Listing E.2, don't use the preceding `&` or trailing `;`. Instead, use only the name of the entity (`someAttribute = "entityName"`).*

**EXAMPLE**

The source document is Listing E.1.

```
<xsl:comment>
  <xsl:value-of
    select="unparsed-entity-uri(advertisement/image/@src)"/>
</xsl:comment>
```

**RESULT TREE FRAGMENT**

logo.png