

## SYBEX Appendix

# Mastering™ XSLT

Chuck White

## Appendix F: XSLT Code Library: Basic Code and Templates

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 F

# XSLT Code Library: Basic Code and Templates

THIS APPENDIX PROVIDES A rather minimal code library of templates that you can use in your code. It certainly isn't a complete library by any means, but if you supplement it with Appendix H, "The Functional Programming Language XSLT: A Proof through Examples" (online at [www.sybex.com](http://www.sybex.com)), which provides a large number of functional processing templates, you'll find a nice set to start out with. (You can find an overview of that article in Appendix C, "An Introduction to Functional Programming with XSLT" at the back of the book.) The code library is broken up into five basic categories:

- ◆ XML code fragments
- ◆ `xmlns:stylesheet` attributes
- ◆ Basic templates
- ◆ HTML
- ◆ Common XSLT tasks

We haven't provided an explanation on how to use them, but if you understand the concepts in this book, you should have no problem with any of them. Some will require more adaptation than others; a few will require very little or none at all. Have fun!

## XML Code Fragments

These are commonly used code fragments you might use in XML source files.

- ◆ XSLT processing instruction:  

```
<?xml-stylesheet type="text/xsl" href="" ?>
```
- ◆ DOCTYPE SYSTEM:  

```
<!DOCTYPE name SYSTEM "" >
```

- ◆ DOCTYPE PUBLIC:  

```
<!DOCTYPE name PUBLIC "" "http://" >
```
- ◆ DOCTYPE internal subset:  

```
<!DOCTYPE name [ ]>
```
- ◆ CDATA section:  

```
<![CDATA[ 'content ' ]]>
```

## ***xsl:stylesheet* Namespace Attributes**

These are commonly used namespaces used with XSLT.

- ◆ XSLT namespace:  

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```
- ◆ XHTML:  

```
xmlns="http://www.w3.org/1999/xhtml"
```
- ◆ MS datatypes namespace:  

```
xmlns:dt='urn:schemas-microsoft-com:datatypes'
```
- ◆ XLink namespace:  

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

## **Basic Templates**

These are commonly used core template constructs.

- ◆ Built-in root and element nodes template:  

```
<xsl:template match="/*">
  <xsl:apply-templates/>
</xsl:template>
```
- ◆ Built-in text template:  

```
<xsl:template match="text(">
  <xsl:value-of select="."/>
</xsl:template>
```
- ◆ Processing instruction and comment template:  

```
<xsl:template match="processing-instruction()|comment()" />
```
- ◆ Identity transform template:  

```
<template>
  <copy>
```

```

        <apply-templates select="@* | * | comment() | processing-instruction() |
        ↳text()"/>
    </copy>
</template>

```

- ◆ Basic root template:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        </xsl:template>
</xsl:stylesheet>

```

## HTML

These are commonly used code fragments used for constructing HTML result trees.

- ◆ XHTML DOCTYPES:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml11-strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml11-
↳transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "DTD/xhtml11-
↳frameset.dtd">

```

In many cases, processors will search the Web for the DTDs shown in the preceding code fragments because the DTDs won't be hard-wired in, so they won't find the DTDs unless you have them in a path relative to your document. So, an alternative is this:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-frameset.dtd">

```

- ◆ Basic HTML template:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method = "html" />

    <xsl:template match="/">
        <html>
            <head></head>
            <body></body>
        </html>
    </xsl:template>
</xsl:stylesheet>

```

- ◆ Two-column table template:

```

<table border="0" width="100%" cellpadding="0" cellspacing="2">

```

```

<xsl:for-each select="">
  <xsl:if test="(position() mod 2) = 1">
    <tr>
      <td width="50%">
        <xsl:value-of select="." />
      </td>
      <td width="50%">
        <!-- If the list ends in odd number -->
        <xsl:if test="following-sibling::">
          <xsl:value-of select="following-sibling::" />
        </xsl:if>
      </td>
    </tr>
  </xsl:if>

  <!-- put an empty row every 5 rows -->
  <xsl:if test="(position() mod 10) = 0">
    <tr><td colspan="2">&#160;</td></tr>
  </xsl:if>
</xsl:for-each>
</table>

```

- ◆ Three-column table template:

```

<table border="1">
  <xsl:for-each select="myElements/myElement">
    <xsl:if test="(position() mod 3) = 1">
      <tr>
        <td><xsl:value-of select="." /></td>
        <td><xsl:value-of select="following-
sibling::myElement[position()=1]" /></td>
        <td><xsl:value-of select="following-
sibling::myElement[position()=2]" /></td>
      </tr>
    </xsl:if>
  </xsl:for-each>
</table>

```

- ◆ Inserting an empty table row every five rows:

```

<xsl:if test="(position() mod 5) = 0">
  <tr>
    <td colspan="2">&#160;</td>
  </tr>
</xsl:if>

```

- ◆ Comma-delimited list of items:

```

<div>
  <xsl:for-each select=" myElements/myElement ">
    <xsl:sort select="myElement" order="descending" />

```

```

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

```

- ◆ Changing element and attribute case:

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" />
  <xsl:variable name="uc" select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ' " />
  <xsl:variable name="lc" select="'abcdefghijklmnopqrstuvwxyz' " />

  <xsl:template match="*">
    <xsl:element name="{translate(name(), $uc, $lc)}">
      <xsl:for-each select="@*">
        <xsl:attribute name="{translate(name(), $uc, $lc)}">
          <xsl:value-of select="." />
        </xsl:attribute>
      </xsl:for-each>
      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>

```

- ◆ Importing XHTML character entities:

```

<!DOCTYPE xsl:stylesheet SYSTEM [
  <!ENTITY xhtml-1at1 SYSTEM
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-1at1.ent">
  <!ENTITY xhtml-special SYSTEM
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-special.ent">
  <!ENTITY xhtml-symbol SYSTEM
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-symbol.ent">
  &xhtml-1at1;
  &xhtml-special;
  &xhtml-symbol;
]>

```

## Common XSLT Tasks

These are stubs for handling common tasks, such as indexing and selecting distinct nodes, that are encountered frequently during XSLT development.

- ◆ Indexed loop:

```

<!-- Use: include this call with the number of iterations
  This sample will loop from 1 to 10 including 1 and 10
-->
<xsl:call-template name="for_loop">
  <xsl:with-param name="i">1</xsl:with-param>

```

```

    <xsl:with-param name="count">10</xsl:with-param>
</xsl:call-template>
<xsl:template name="for_loop">
  <xsl:param name="i"/>
  <xsl:param name="count"/>
  <xsl:if test="$i &lt;= $count">
<!-- body of loop goes here -->
  </xsl:if>
  <xsl:if test="$i &lt;= $count">
    <xsl:call-template name="for_loop">
      <xsl:with-param name="i">
        <!-- Increment index-->
        <xsl:value-of select="$i + 1"/>
      </xsl:with-param>
      <xsl:with-param name="count">
        <xsl:value-of select="$count"/>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

```

◆ **xsl:key: Selecting DISTINCT, v.1:**

```

<!-- the key declaration must be a top-level element -->
<!-- substitute myKey, myElement and @a with your own key name
and attribute name -->
  <xsl:key name="myKey" match="@a" use="."/>

  <xsl:template name="myKey">
    <table border="0" cellPadding="1" cellSpacing="1" width="100%">
      <xsl:for-each select="/*/myElement[count(@a | key('myKey', @a)[1])
        => 1]">
        <tr>
          <xsl:value-of select="@a" />
        </tr>
      </xsl:for-each>

```

◆ **xsl:key: Selecting DISTINCT, v.2:**

```

<!-- the key declaration must be a top-level element -->
<xsl:key name="keyname" match="@att" use="."/>

<!--
  substitute foot with XPath to the node you want to iterate
  substitute @a with the name of the attribute or . for txt node.
-->
<xsl:for-each select="xpath[count(@a | key('keyname', @a)[1]) = 1]">
  <xsl:value-of select="@a" />
</xsl:for-each>

```

- ◆ Renaming elements:

```
<template match="oldName">
  <element name="newName">
    <apply-templates select="@* | * | comment() | processing-instruction() |
      ↳text()" />
  </element>
</template>
```

- ◆ Copy elements and their attributes:

```
<xsl:template match="xsl:*">
  <xsl:copy>
    <xsl:for-each select="@*">
      <xsl:copy><xsl:value-of select="."/;></xsl:copy>
    </xsl:for-each>
    <xsl:apply-templates select="node()" />
  </xsl:copy>
</xsl:template>
```

- ◆ Copy attributes:

```
<xsl:for-each select = "@*">
  <xsl:copy />
</xsl:for-each>
```

- ◆ Copy text, comments, and PIs:

```
<xsl:template match="comment() | processing-instruction() | text()">
  <xsl:copy>
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>
```

- ◆ Testing uniqueness of nodes using Muenchian Method and `count()`:

In the following code fragment, `keyName` is the name of the key, and `keyUse` matches the use attribute of the defined key.

```
contact[count(. | key('keyName', keyUse)[1]) = 1]
```

- ◆ Testing uniqueness of nodes using Muenchian Method and `generate-id()`:

In the following code fragment, `keyName` is the name of the key, and `keyUse` matches the use attribute of the defined key.

```
contact[generate-id() =
  generate-id(key('keyName', keyUse)[1])]
```

- ◆ MSXSL script tag:

This is the MSXSL Script tag used for managing JavaScript functions within stylesheets targeted for the MSXML-based processor.

```
<msxsl:script language="VBScript" implements-prefix=""></msxsl:script>
```