Top-Level Declarations

<**xsl:attribute-set name** = qname
use-attribute-sets = qnames>
xsl:attribute*
</**xsl:attribute-set**>

<**xsl:character-map name** = qname
use-character-maps = qnames>
xsl:output-character*
<**xsl:output-character character** = char
string = string />
</**xsl:character-map**>

One or more **xsl:output-character** is allowed.

<**xsl:decimal-format** name = qname
decimal-separator = char
grouping-separator = char
infinity = string
minus-sign = char
NaN = string
percent = char
per-mille = char
zero-digit = char
digit = char
pattern-separator = char />

<**xsl:function name** = qname
as = sequence-type
override = "yes" | "no">
xsl:param*, sequence-constructor
</**xsl:function**>

<**xsl:import-schema** namespace = uri
schema-location = uri>
xs:schema?
</**xsl:import-schema**>

<**xsl:include href** = uri />

<**xsl:key name** = qname
match = pattern
use = expression
collation = uri>
sequence-constructor
</**xsl:key**>

<**xsl:namespace-alias**
stylesheet-prefix = prefix | "#default"
result-prefix = prefix | "#default" />

Content Specification Options

?	optional
*	zero or more
+	one or more
#PCDATA	just text
sequence-constructor	Instructions and text

<**xsl:output** name = qname
method = "xml" | "html" | "xhtml" |
"text" |qname-but-not-ncname
byte-order-mark = "yes" | "no"
cdata-section-elements = qnames
doctype-public = string
doctype-system = string
encoding = string
escape-uri-attributes = "yes" | "no"
include-content-type = "yes" | "no"
indent = "yes" | "no"
media-type = string
normalization-form = "NFC" | "NFD" |
"NFKC" | "NFKD" | "none" |
"fully-normalized" | nmtoken
omit-xml-declaration = "yes" | "no"
standalone = "yes" | "no" | "omit"
undeclare-prefixes = "yes" | "no"
use-character-maps = qnames
version = nmtoken />

<**xsl:param name** = qname
select = expression
as = sequence-type
required = "yes" | "no"
tunnel = "yes" | "no">
sequence-constructor
</**xsl:param**>

xsl:param is also allowed in **xsl:function** and **xsl:template**.

<**xsl:preserve-space elements** = tokens />

<**xsl:strip-space elements** = tokens />

<**xsl:template** match = pattern
name = qname
priority = number
mode = tokens
as = sequence-type>
xsl:param*, sequence-constructor
</**xsl:template**>

<**xsl:variable name** = qname
select = expression
as = sequence-type>
sequence-constructor
</**xsl:variable**>

xsl:variable is also allowed in sequence-constructor contexts.

Attribute Specification Options

{ }	specified using an attribute value template
bold =	required attribute
non-bold =	optional attribute

Node Constructing Instructions

<**xsl:attribute name** = { qname }
namespace = { uri }
select = expression
separator = { string }
type = qname
validation = "strict" | "lax" |
"preserve" | "strip">
sequence-constructor
</**xsl:attribute**>

<**xsl:comment** select = expression>
sequence-constructor
</**xsl:comment**>

<**xsl:document** type = qname
validation = "strict" | "lax" |
"preserve" | "strip" >
sequence-constructor
</**xsl:document**>

<**xsl:element name** = { qname }
namespace = { uri}
inherit-namespaces = "yes" | "no"
use-attribute-sets = qnames
type = qname
validation = "strict" | "lax" |
"preserve" | "strip">
sequence-constructor
</**xsl:element**>

Element nodes can also be constructed using XML elements not in the **xsl**: namespace, which can also specify **xsl:type**, **xsl:validation** and **xsl:use-attribute-sets** attributes.

<**xsl:namespace name** = { ncname }
select = expression>
sequence-constructor
</**xsl:namespace**>

<**xsl:processing-instruction**
name = { ncname }
select = expression>
sequence-constructor
</**xsl:processing-instruction**>

<**xsl:sequence select** = expression>
xsl:fallback*
</**xsl:sequence**>

<**xsl:text** disable-output-escaping = "yes" | "no" >
#PCDATA
</**xsl:text**>

disable-output-escaping is deprecated.

Text also constructs text nodes.

XSL-List:
http://www.mulberrytech.com/xsl/xsl-list

<**xsl:result-document** format = { qname }
href = { uri }
validation = "strict" | "lax" |
"preserve" | "strip"
type = qname
method = { "xml" | "html" | "xhtml" |
"text" | qname-but-not-ncname }
byte-order-mark = { "yes" | "no" }
cdata-section-elements = { qnames }
doctype-public = { string }
doctype-system = { string }
encoding = { string }
escape-uri-attributes = { "yes" | "no" }
include-content-type = { "yes" | "no" }
indent = { "yes" | "no" }
media-type = { string }
normalization-form = { "NFC" | "NFD" |
"NFKC" | "NFKD" | "none" |
"fully-normalized" | nmtoken }
omit-xml-declaration = { "yes" | "no" }
standalone = { "yes" | "no" | "omit" }
undeclare-prefixes = { "yes" | "no" }
use-character-maps = qnames
output-version = { nmtoken } >
sequence-constructor
</**xsl:result-document**>

Allowed Attribute Values:

char	a single character
expression	an XPath expression
id	an ID attribute value
ncname	a name with no namespace prefix
nmtoken	a number token
number	a number (only digits)
pattern	an XPath expression conforming to pattern syntax
prefix	a namespace prefix
qname-but-not-ncname	a name with a namespace prefix
qname	a name with or without a namespace prefix
sequence-type	an XML Schema sequence type (with *)
string	just text
token	specific to its use
uri-list	white-space separated list of URIs
uri	a uniform resource identifier

Conditional and Looping Instructions

```
<xsl:analyze-string select = expression
        regex = { string }
        flags = { string }>
    <xsl:matching-substring>
        sequence-constructor
        </xsl:matching-substring>
    <xsl:non-matching-substring>
        sequence-constructor
        </xsl:non-matching-substring>
    xsl:fallback*
    </xsl:analyze-string>
```

One but not both of **xsl:matching-substring** and **xsl:non-matching-substring** can be omitted.

regex-group(N) returns the **N**th group matched by the **regex** within **xsl:matching-substring**.

```
<xsl:choose>
    <xsl:when test = expression>
        sequence-constructor
        </xsl:when>
    <xsl:otherwise>
        sequence-constructor
        </xsl:otherwise>
    </xsl:choose>
```

One or more **xsl:when** and zero or one **xsl:otherwise** are alllowed.

```
<xsl:for-each select = expression>
    xsl:sort*,sequence-constructor
    </xsl:for-each>
```

```
<xsl:for-each-group select = expression
        group-by = expression
        group-adjacent = expression
        group-starting-with = pattern
        group-ending-with = pattern
        collation = { uri }>
    xsl:sort*,sequence-constructor
    </xsl:for-each-group>
```

```
<xsl:if test = expression>
    sequence-constructor
    </xsl:if>
```

Standard Attributes

Standard attributes are allowed on all elements. When not on **xsl:** elements, the **xsl:** prefix is required on the attribute name.

```
    [xsl:]default-collation = uri
    [xsl:]exclude-result-prefixes = tokens
    [xsl:]extension-element-prefixes = tokens
    [xsl:]use-when = expression
    [xsl:]version = "1.0" | "2.0"
    [xsl:]xpath-default-namespace = uri
```

Value/Copy Instructions

```
<xsl:copy copy-namespaces = "yes" | "no"
        inherit-namespaces = "yes" | "no"
        use-attribute-sets = qnames
        type = qname
        validation = "strict" | "lax" |
                     "preserve" | "strip">
    sequence-constructor
    </xsl:copy>
```

```
<xsl:copy-of select = expression
        copy-namespaces = "yes" | "no"
        type = qname
        validation = "strict" | "lax" |
                     "preserve" | "strip" />
```

```
<xsl:number value = expression
        select = expression
        level = "single" | "multiple" | "any"
        count = pattern
        from = pattern
        format = { string }
        lang = { nmtoken }
        letter-value = { "alphabetic" |
                         "traditional" }
        ordinal = { string }
        grouping-separator = { char }
        grouping-size = { number } />
```

```
<xsl:perform-sort select = expression>
    xsl:sort+, sequence-constructor
    </xsl:perform-sort>
```

```
<xsl:value-of select = expression
        separator = { string }
        disable-output-escaping = "yes" | "no" >
    sequence-constructor
    </xsl:value-of>
```

disable-output-escaping is deprecated.

```
<xsl:sort select = expression
        lang = { nmtoken }
        order = { "ascending" | "descending"}
        collation = { uri }
        stable = { "yes" | "no" }
        case-order = { "upper-first" | "lower-first" }
        data-type = { "text" | "number" |
                      qname-but-not-ncname } >
    sequence-constructor
    </xsl:sort>
```

xsl:sort is used in **xsl:for-each**, **xsl:for-each-group**, **xsl:apply-templates** and **xsl:perform-sort**.

XSLT 2.0:
http://www.w3.org/TR/xslt20/

XPath 2.0:
http://www.w3.org/TR/xpath20/

2008-07-21

XSLT 2.0 Quick Reference

Sam Wilmott
sam@wilmott.ca
http://www.wilmott.ca

and

Mulberry Technologies, Inc.
17 West Jefferson Street, Suite 207
Rockville, MD 20850 USA
Phone: +1 301/315-9631
Fax: +1 301/315-8285
info@mulberrytech.com
http://www.mulberrytech.com

Mulberry Technologies, Inc.

The Stylesheet Element

```
<xsl:stylesheet id = id
        extension-element-prefixes = tokens
        exclude-result-prefixes = tokens
        version = "1.0" | "2.0"
        xpath-default-namespace = uri
        default-validation = "preserve" | "strip"
        default-collation = uri-list
        input-type-annotations = "preserve" |
                                 "strip" | "unspecified"
        xmlns:xsl=
          "http://www.w3.org/1999/XSL/Transform">
    xsl:import*, top-level-declarations
    </xsl:stylesheet>
```

xsl:transform is a synonym for **xsl:stylesheet**.

```
<xsl:import href = uri />
```

A literal result element can be used in place of **xsl:stylesheet**, so long as it specifies attribute **xsl:version** and namespace **xmlns:xsl**.

Template Invocation Instructions

```
<xsl:apply-imports>
    xsl:with-param*
    </xsl:apply-imports>
```

```
<xsl:apply-templates select = expression
        mode = token>
    (xsl:sort | xsl:with-param)*
    </xsl:apply-templates>
```

```
<xsl:call-template name = qname>
    xsl:with-param*
    </xsl:call-template>
```

```
<xsl:next-match>
    (xsl:with-param | xsl:fallback)*
    </xsl:next-match>
```

```
<xsl:with-param name = qname
        select = expression
        as = sequence-type
        tunnel = "yes" | "no">
    sequence-constructor
    </xsl:with-param>
```

Exception-Handling Instructions

```
<xsl:fallback>
    sequence-constructor
    </xsl:fallback>
```

```
<xsl:message select = expression
        terminate = { "yes" | "no" }>
    sequence-constructor
    </xsl:message>
```