Location Paths [XPath §2]

Optional '/', zero or more **location steps**, separated by '/'

Location Steps [XPath §2.1]

Axis specifier, **node test**, zero or more **predicates**

Axis Specifiers [XPath §2.2]

ancestor::
ancestor-or-self::
attribute::
child::
descendant::
descendant-or-self::
following::
following-sibling::
namespace::
parent::
preceding::
preceding-sibling::
self::

Node Tests [XPath §2.3]

name
URI:*name*
prefix:*name*
*
prefix:*
node()
text()
comment()
processing-instruction()
processing-instruction(*literal*)

Abbreviated Syntax for Location Paths

(nothing)	child::
@	attribute::
//	/descendant-or-self::node()/
.	self::node()
..	parent::node()
/	Node tree root

Predicate [XPath §2.4]

[*expr*]

Variable Reference [XPath §3.7]

$*qname*

Literal Result Elements [§7.1.1]

Any element not in the xsl: namespace and not an extension element

XSLT

http://www.w3.org/TR/xslt

XPath

http://www.w3.org/TR/xpath

XSL-List

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

XPath Operators

Parentheses may be used for grouping.

Node-sets [XPath §3.3]

| [*expr*] / //

Booleans [XPath §3.4]

<=, <, >=, > **=, !=** **and** **or**

Numbers [XPath §3.5]

-*expr* ***, div, mod** **+, -**

XPath Core Function Library

Node Set Functions [XPath §4.1]

number **last**()
number **position**()
number **count**(*node-set*)
node-set **id**(*object*)
string **local-name**(*node-set*?)
string **namespace-uri**(*node-set*?)
string **name**(*node-set*?)

String Functions [XPath §4.2]

string **string**(*object*?)
string **concat**(*string*, *string*, *string**)
boolean **starts-with**(*string*, *string*)
boolean **contains**(*string*, *string*)
string **substring-before**(*string*, *string*)
string **substring-after**(*string*, *string*)
string **substring**(*string*, *number*, *number*?)
number **string-length**(*string*?)
string **normalize-space**(*string*?)
string **translate**(*string*, *string*, *string*)

Boolean Functions [XPath §4.3]

boolean **boolean**(*object*)
boolean **not**(*object*)
boolean **true**()
boolean **false**()
boolean **lang**(*string*)

Number Functions [XPath §4.4]

number **number**(*object*?)
number **sum**(*node-set*)
number **floor**(*number*)
number **ceiling**(*number*)
number **round**(*number*)

XSLT and XPath Quick Reference

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

XSLT Functions [§12, §15]

node-set **document**(*object*, *node-set*?)
node-set **key**(*string*, *object*)
string **format-number**(*number*, *string*, *string*?)
node-set **current**()
string **unparsed-entity-uri**(*string*)
string **generate-id**(*node-set*?)
object **system-property**(*string*)
boolean **element-available**(*string*)
boolean **function-available**(*string*)

Node Types [XPath §5]

Root
Element
Attribute
Namespace
Processing Instruction
Comment
Text

Object Types [§11.1, XPath §1]

boolean	True or false
number	Floating-point number
string	UCS characters
node-set	Set of nodes selected by a path
Result tree fragment	XSLT only. Fragment of the result tree

Expression Context [§4, XPath §1]

Context node (a node)
Context position (a number)
Context size (a number)
Variable bindings in scope
Namespace declarations in scope
Function library

Built-in Template Rules [§5.8]

```
<xsl:template match="*|/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*|/" mode="m">
    <xsl:apply-templates mode="m"/>
</xsl:template>

<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>

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

Built-in template rule for namespaces is to do nothing

XSLT Elements

Stylesheet Element [§2.2]

```
<xsl:stylesheet version="1.0" id="id"
    extension-element-prefixes="tokens"
    exclude-result-prefixes="tokens"
    xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform"> xsl:import*, top-level elements
</xsl:stylesheet>
```

xsl:transform is a synonym for xsl:stylesheet

Combining Stylesheets [§2.6]

```
<xsl:include href="uri-reference"/>

<xsl:import href="uri-reference"/>
```

Whitespace Stripping [§3.4]

```
<xsl:strip-space elements="tokens"/>

<xsl:preserve-space elements="tokens"/>
```

Defining Template Rules [§5.3]

```
<xsl:template match="pattern" name="qname"
    priority="number" mode="qname">
    xsl:param* followed by text, literal result elements
    and/or XSL elements </xsl:template>
```

Applying Template Rules [§5.4]

```
<xsl:apply-templates select="node-set-exp"
    mode="qname"/>
<xsl:apply-templates select="node-set-exp"
    mode="qname">
    (xsl:sort | xsl:with-param)* </xsl:apply-templates>
```

Overriding Template Rules [§5.6]

```
<xsl:apply-imports/>
```

Named Templates [§6]

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

Namespace Alias [§7.1.1]

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

Creating Elements [§7.1.2]

```
<xsl:element name="{qname}"
    namespace="{uri-reference}"
    use-attribute-sets="qnames">...</xsl:element>
```

Creating Attributes [§7.1.3]

```
<xsl:attribute name="{qname}"
    namespace="{uri-reference}">...</xsl:attribute>
```

Named Attribute Sets [§7.1.4]

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

Creating Text [§7.2]

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

Processing Instructions [§7.3]

```
<xsl:processing-instruction name="{ncname}">
    ...</xsl:processing-instruction>
```

Creating Comments [§7.4]

```
<xsl:comment>...</xsl:comment>
```

Copying [§7.5]

```
<xsl:copy use-attribute-sets="qnames">
    ...</xsl:copy>
```

Generating Text [§7.6.1]

```
<xsl:value-of select="string-expr"
    disable-output-escaping="yes|no"/>
```

Attribute Value Templates [§7.6.2]

```
<element attribute="{expr}"/>
```

Numbering [§7.7]

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

Repetition [§8]

```
<xsl:for-each select="node-set-expr">
    xsl:sort*, ...</xsl:for-each>
```

Conditional Processing [§9]

```
<xsl:if test="boolean-expr">...</xsl:if>

<xsl:choose>
    <xsl:when test="expr">...</xsl:when>+
    <xsl:otherwise>...</xsl:otherwise>?
</xsl:choose>
```

Sorting [§10]

```
<xsl:sort select="string-expr" lang="{nmtoken}"
    data-type="{text|number|qname-but-not-
    ncname}" order="{ascending|descending}"
    case-order="{upper-first|lower-first}"/>
```

Variables and Parameters [§11]

```
<xsl:variable name="qname" select="expr"/>
<xsl:variable name="qname">...</xsl:variable>

<xsl:param name="qname" select="expr"/>
<xsl:param name="qname">...</xsl:param>
```

Using Values [§11.3]

```
<xsl:copy-of select="expr"/>
```

Passing Parameters [§11.6]

```
<xsl:with-param name="expr" select="expr"/>
<xsl:with-param name="expr">...</xsl:with-param>
```

Keys [§12.2]

```
<xsl:key name="qname" match="pattern"
    use="expr"/>
```

Number Formatting [§12.3]

```
<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"/>
```

Messages [§13]

```
<xsl:message terminate="yes|no">
    ...</xsl:message>
```

Fallback [§15]

```
<xsl:fallback>...</xsl:fallback>
```

Output [§16]

```
<xsl:output
    method="xml|html|text|qname-but-not-ncname"
    version="nmtoken" encoding="string"
    omit-xml-declaration="yes|no"
    doctype-public="string" doctype-system="string"
    standalone="yes|no" indent="yes|no"
    cdata-section-elements="qnames"
    media-type="string"/>
```

Key

xsl:stylesheet	Element
version=	Required attribute
version=	Optional attribute
{expr}	Attribute value template. Text between any { and } is evaluated as an expression. Attribute value must evaluate to indicated attribute type.
...	Anything allowed in a template
|	Separates alternative values
?	Zero or one occurrences
*	Zero or more occurrences
+	One or more occurrences
#PCDATA	Character data

Attribute Value Types

1.0	Literal value
boolean-expr	Expression returning boolean value
char	Single character
expr	Expression
id	XML name used as identifier
ncname	XML name not containing a colon (:)
node-set-expr	Expression returning a node set
number-expr	Expression returning a number
pattern	XSLT pattern
prefix	Namespace prefix
qname	Namespace-qualified XML name comprising local part and optional prefix
qname-but-not-ncname	Namespace-qualified name comprising local part and prefix
token	Meaning varies with context. See Rec.
uri-reference	Reference to Universal Resource Identifier

XSLT 2.0 Quick reference. 2007-03-18Z

http://www.dpawson.co.uk/xsl/rev2/rev2.html
Produced with DiType from RenderX

There are a number of standard attributes that may appear on any XSLT element: specifically version, exclude-result-prefixes, extension-element-prefixes, xpath-default-namespace, default-collation, and use-when.

Element xsl:analyze-string
Attributes:
- select as *expression*
- regex{ as *string* }
- flags{ as *string* }

<--Content:(xsl:matching-substring?, xsl:non-matching-substring?, xsl:fallback*)-->

Element xsl:apply-imports
<--Content:(xsl:with-param*)-->

Element xsl:apply-templates
Attributes:
- select as *expression*
- mode as *token*

<--Content:(xsl:sort | xsl:with-param)* -->

Element xsl:attribute
(*sequence-constructor*)
Attributes:
- name{ as *qname* }
- namespace{ as *uri-reference* }
- select as *expression*
- separator{ as *string* }
- type as *qname*
- validation "strict| lax| preserve| strip"

<--Content:(sequence constructor)-->

Element xsl:attribute-set
Attributes:
- name as *qname*
- use-attribute-sets as *qnames*

<--Content:(xsl:attribute*)-->

Element xsl:call-template
Attributes:
- name as *qname*

<--Content:(xsl:with-param*)-->

Element xsl:character-map
Attributes:
- name as *qname*
- use-character-maps as *qnames*

<--Content:(xsl:output-character*)-->

Element xsl:choose
<--Content:(xsl:when+, xsl:otherwise?)-->

Element xsl:comment
(*sequence-constructor*)
Attributes:
- select as *expression*

<--Content:(sequence constructor)-->

Element xsl:copy
(*sequence-constructor*)
Attributes:
- copy-namespaces "yes| no"
- inherit-namespaces "yes| no"
- use-attribute-sets as *qnames*
- type as *qname*
- validation "strict| lax| preserve| strip"

<--Content:(sequence constructor)-->

Element xsl:copy-of
Attributes:
- select as *expression*
- copy-namespaces "yes| no"
- type as *qname*
- validation "strict| lax| preserve| strip"

Element xsl:decimal-format
Attributes:
- name as *qname*
- decimal-separator as *char*
- grouping-separator as *char*
- infinity as *string*
- minus-sign as *char*
- NaN as *string*
- percent as *char*
- per-mille as *char*
- zero-digit as *char*
- digit as *char*
- pattern-separator as *char*

Element xsl:document
(*sequence-constructor*)
Attributes:
- validation "strict| lax| preserve| strip"
- type as *qname*

<--Content:(sequence constructor)-->

Element xsl:element
(*sequence-constructor*)
Attributes:
- name{ as *qname* }
- namespace{ as *uri-reference* }
- inherit-namespaces "yes| no"
- use-attribute-sets as *qnames*
- type as *qname*
- validation "strict| lax| preserve| strip"

<--Content:(sequence constructor)-->

Element xsl:fallback
(*sequence-constructor*) <--Content:(sequence constructor)-->

Element xsl:for-each
Attributes:
- select as *expression*

<--Content:(xsl:sort*,   sequence constructor)-->

Element xsl:for-each-group
Attributes:
- select as *expression*
- group-by as *expression*
- group-adjacent as *expression*
- group-starting-with as *pattern*
- group-ending-with as *pattern*
- collation{ as *uri* }

<--Content:(xsl:sort*,   sequence constructor)-->

Element xsl:function
Attributes:
- name as *qname*
- as as *sequence-type*
- override "yes| no"

<--Content:(xsl:param*,   sequence constructor)-->

Element xsl:if
(*sequence-constructor*)
Attributes:
- test as *expression*

<--Content:(sequence constructor)-->

Element xsl:import
Attributes:
- href as *uri-reference*

Element xsl:import-schema
Attributes:
- namespace as *uri-reference*
- schema-location as *uri-reference*

<--Content:(xsl:xs:schema?)-->

Element xsl:include
Attributes:
- href as *uri-reference*

Element xsl:key
(*sequence-constructor*)
Attributes:
- name as *qname*
- match as *pattern*
- use as *expression*
- collation as *uri*

<--Content:(sequence constructor)-->

Element xsl:matching-substring
(*sequence-constructor*) <--Content:(sequence constructor)-->

Element xsl:message
(*sequence-constructor*)
Attributes:

- select as *expression*
- terminate{ "yes| no" }

<--Content:(sequence constructor)-->

Element xsl:namespace
(*sequence-constructor*)
Attributes:

- name{ as *ncname* }
- select as *expression*

<--Content:(sequence constructor)-->

Element xsl:namespace-alias
Attributes:

- stylesheet-prefix as *prefix* "#default"
- result-prefix as *prefix* "#default"

Element xsl:next-match
<--Content:(xsl:with-param | xsl:fallback)* -->

Element xsl:non-matching-substring
(*sequence-constructor*) <--Content:(sequence constructor)-->

Element xsl:number
Attributes:

- value as *expression*
- select as *expression*
- level "single| multiple| any"
- count as *pattern*
- from as *pattern*
- format{ as *string* }
- lang{ as *nmtoken* }
- letter-value{ "alphabetic| traditional" }
- ordinal{ as *string* }
- grouping-separator{ as *char* }
- grouping-size{ as *number* }

Element xsl:otherwise
(*sequence-constructor*) <--Content:(sequence constructor)-->

Element xsl:output
Attributes:

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

Element xsl:output-character
Attributes:

- character as *char*
- string as *string*

Element xsl:param
(*sequence-constructor*)
Attributes:

- name as *qname*
- select as *expression*
- as as *sequence-type*
- required "yes| no"
- tunnel "yes| no"

<--Content:(sequence constructor)-->

Element xsl:perform-sort
Attributes:

- select as *expression*

<--Content:(xsl:sort+,   sequence constructor)-->

Element xsl:preserve-space
Attributes:

- elements as *tokens*

Element xsl:processing-instruction
(*sequence-constructor*)
Attributes:

- name{ as *ncname* }
- select as *expression*

<--Content:(sequence constructor)-->

Element xsl:result-document
(*sequence-constructor*)
Attributes:

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

<--Content:(sequence constructor)-->

Element xsl:sequence
Attributes:

- select as *expression*

<--Content:(xsl:fallback*)-->

Element xsl:sort
(*sequence-constructor*)
Attributes:

- select as *expression*
- lang{ as *nmtoken* }
- order{ "ascending| descending" }
- collation{ as *uri* }
- stable{ "yes| no" }
- case-order{ "upper-first| lower-first" }
- data-type{ "text| number" as *qname-but-not-ncname* }

<--Content:(sequence constructor)-->

Element xsl:strip-space
Attributes:

- elements as *tokens*

Element xsl:stylesheet
Attributes:

- id as *id*
- extension-element-prefixes as *tokens*
- exclude-result-prefixes as *tokens*
- version as *number*
- xpath-default-namespace as *uri*
- default-validation "preserve| strip"
- default-collation as *uri-list*
- input-type-annotations "preserve| strip| unspecified"

<--Content:(xsl:import*,   other declarations)-->

Element xsl:template
Attributes:

- match as *pattern*
- name as *qname*
- priority as *number*
- mode as *tokens*
- as as *sequence-type*

<--Content:(xsl:param*,   sequence constructor)-->

Element xsl:text
Attributes:

- disable-output-escaping "yes| no" *Deprecated*

<--Content:(<text/>)-->

Element xsl:transform
Attributes:

- id as *id*
- extension-element-prefixes as *tokens*
- exclude-result-prefixes as *tokens*
- version as *number*
- xpath-default-namespace as *uri*
- default-validation "preserve| strip"
- default-collation as *uri-list*
- input-type-annotations "preserve| strip| unspecified"

<--Content:(xsl:import*,   other declarations)-->

Element xsl:value-of
(*sequence-constructor*)
Attributes:

- select as *expression*
- separator{ as *string* }
- disable-output-escaping "yes| no" *Deprecated*

<--Content:(sequence constructor)-->

Element xsl:variable
(*sequence-constructor*)
Attributes:

- name as *qname*
- select as *expression*
- as as *sequence-type*

<--Content:(sequence constructor)-->

Element xsl:when
(*sequence-constructor*)
Attributes:

- test as *expression*

<--Content:(sequence constructor)-->

Element xsl:with-param
(*sequence-constructor*)
Attributes:

- name as *qname*
- select as *expression*
- as as *sequence-type*
- tunnel "yes| no"

<--Content:(sequence constructor)-->

XSLT functions

xslt: **current** () as *item()*
xslt: **current-group** () as *item()*
xslt: **current-grouping-key** () as *xs:anyAtomicType*
xslt: **document** ($uri-sequence as item() [$base-node] as node()) as *node()*
xslt: **element-available** ($element-name as xs:string) as *xs:boolean*
xslt: **format-date** ($value as xs:date, $picture as xs:string, $language as xs:string, $calendar as xs:string, $country as xs:string) as *xs:string*
xslt: **format-dateTime** ($value as xs:dateTime, $picture as xs:string, $language as xs:string, $calendar as xs:string, $country as xs:string) as *xs:string*
xslt: **format-number** ($value as numeric, $picture as xs:string [$decimal-format-name] as xs:string) as *xs:string*
xslt: **format-time** ($value as xs:time, $picture as xs:string, $language as xs:string, $calendar as xs:string, $country as xs:string) as *xs:string*
xslt: **function-available** ($function-name as xs:string [$arity] as xs:integer) as *xs:boolean*
xslt: **generate-id** ([$node] as node()) as *xs:string*
xslt: **key** ($key-name as xs:string, $key-value as xs:anyAtomicType [$top] as node()) as *node()*
xslt: **regex-group** ($group-number as xs:integer) as *xs:string*
xslt: **system-property** ($property-name as xs:string) as *xs:string*
xslt: **type-available** ($type-name as xs:string) as *xs:boolean*
xslt: **unparsed-entity-public-id** ($entity-name as xs:string) as *xs:string*
xslt: **unparsed-entity-uri** ($entity-name as xs:string) as *xs:anyURI*
xslt: **unparsed-text** ($href as xs:string [$encoding] as xs:string) as *xs:string*
xslt: **unparsed-text-available** ($href as xs:string [$encoding] as xs:string) as *xs:boolean*

XPATH functions

xpath: **ENTITY** ($arg as xs:anyAtomicType) as *xs:ENTITY*
xpath: **ID** ($arg as xs:anyAtomicType) as *xs:ID*
xpath: **IDREF** ($arg as xs:anyAtomicType) as *xs:IDREF*
xpath: **NCName** ($arg as xs:anyAtomicType) as *xs:NCName*
xpath: **NMTOKEN** ($arg as xs:anyAtomicType) as *xs:NMTOKEN*
xpath: **Name** ($arg as xs:anyAtomicType) as *xs:Name*
xpath: **QName** ($arg as xs:anyAtomicType [$paramURI] as xs:string, [$paramQName] as xs:string) as *xs:QName*
xpath: **abs** ($arg as numeric) as *numeric*
xpath: **adjust-date-to-timezone** ($arg as xs:date [$timezone] as xs:dayTimeDuration) as *xs:date*
xpath: **adjust-dateTime-to-timezone** ($arg as xs:dateTime [$timezone] as xs:dayTimeDuration) as *xs:dateTime*
xpath: **adjust-time-to-timezone** ($arg as xs:time [$timezone] as xs:dayTimeDuration) as *xs:time*
xpath: **anyURI** ($arg as xs:anyAtomicType) as *xs:anyURI*
xpath: **avg** ($arg as xs:anyAtomicType*) as *xs:anyAtomicType*
xpath: **base-uri** ([$arg] as node()) as *xs:anyURI*
xpath: **base64Binary** ($arg as xs:anyAtomicType) as *xs:base64Binary*
xpath: **boolean** ($arg as xs:anyAtomicType) as *xs:boolean*
xpath: **byte** ($arg as xs:anyAtomicType) as *xs:byte*
xpath: **ceiling** ($arg as numeric) as *numeric*
xpath: **codepoint-equal** ($comparand1 as xs:string, $comparand2 as xs:string) as *xs:boolean*
xpath: **codepoints-to-string** ($arg as xs:integer*) as *xs:string*
xpath: **collection** ([$arg] as xs:string) as *node()**
xpath: **compare** ($comparand1 as xs:string, $comparand2 as xs:string [$collation] as xs:string) as *xs:integer*
xpath: **concat** ($arg1 as xs:anyAtomicType, $arg2 as xs:anyAtomicType, $... as) as *xs:string*
xpath: **contains** ($arg1 as xs:string, $arg2 as xs:string [$collation] as xs:string) as *xs:boolean*
xpath: **count** ($arg as item()*) as *xs:integer*
xpath: **current-date** () as *xs:date*
xpath: **current-dateTime** () as *xs:dateTime*
xpath: **current-time** () as *xs:time*
xpath: **data** ($arg as item()*) as *xs:anyAtomicType**
xpath: **date** ($arg as xs:anyAtomicType) as *xs:date*
xpath: **dateTime** ($arg as xs:anyAtomicType [$arg1] as xs:date, [$arg2] as xs:time) as *xs:dateTime*
xpath: **day-from-date** ($arg as xs:date) as *xs:integer*
xpath: **day-from-dateTime** ($arg as xs:dateTime) as *xs:integer*
xpath: **dayTimeDuration** ($arg as xs:anyAtomicType) as *xs:dayTimeDuration*
xpath: **days-from-duration** ($arg as xs:duration) as *xs:integer*
xpath: **decimal** ($arg as xs:anyAtomicType) as *xs:decimal*
xpath: **deep-equal** ($parameter1 as item()*, $parameter2 as item()* [$collation] as string) as *xs:boolean*
xpath: **default-collation** () as *xs:string*
xpath: **distinct-values** ($arg as xs:anyAtomicType* [$collation] as xs:string) as *xs:anyAtomicType**
xpath: **doc** ($uri as xs:string) as *document-node()*
xpath: **doc-available** ($uri as xs:string) as *xs:boolean*
xpath: **document-uri** ($arg as node()) as *xs:anyURI*
xpath: **double** ($arg as xs:anyAtomicType) as *xs:double*
xpath: **duration** ($arg as xs:anyAtomicType) as *xs:duration*
xpath: **empty** ($arg as item()*) as *xs:boolean*
xpath: **encode-for-uri** ($uri-part as xs:string) as *xs:string*
xpath: **ends-with** ($arg1 as xs:string, $arg2 as xs:string [$collation] as xs:string) as *xs:boolean*
xpath: **error** ([$error] as xs:QName [$error] as xs:QName, [$description] as xs:string [$error] as xs:QName, [$description] as xs:string, [$error-object] as item()*) as *none*
xpath: **escape-html-uri** ($uri as xs:string) as *xs:string*
xpath: **exactly-one** ($arg as item()*) as *item()*

xpath: **exists** ($arg as item()*) as _xs:boolean_
xpath: **false** () as _xs:boolean_
xpath: **float** ($arg as xs:anyAtomicType) as _xs:float_
xpath: **floor** ($arg as numeric) as _numeric_
xpath: **gDay** ($arg as xs:anyAtomicType) as _xs:gDay_
xpath: **gMonth** ($arg as xs:anyAtomicType) as _xs:gMonth_
xpath: **gMonthDay** ($arg as xs:anyAtomicType) as _xs:gMonthDay_
xpath: **gYear** ($arg as xs:anyAtomicType) as _xs:gYear_
xpath: **gYearMonth** ($arg as xs:anyAtomicType) as _xs:gYearMonth_
xpath: **hexBinary** ($arg as xs:anyAtomicType) as _xs:hexBinary_
xpath: **hours-from-dateTime** ($arg as xs:dateTime) as _xs:integer_
xpath: **hours-from-duration** ($arg as xs:duration) as _xs:integer_
xpath: **hours-from-time** ($arg as xs:time) as _xs:integer_
xpath: **id** ($arg as xs:string* [$node] as node()) as _element()*_
xpath: **idref** ($arg as xs:string* [$node] as node()) as _node()*_
xpath: **implicit-timezone** () as _xs:dayTimeDuration_
xpath: **in-scope-prefixes** ($element as element()) as _xs:string*_
xpath: **index-of** ($seqParam as xs:anyAtomicType*, $srchParam as xs:anyAtomicType [$collation] as xs:string) as _xs:integer*_
xpath: **insert-before** ($target as item()*, $position as xs:integer, $inserts as item()*) as _item()*_
xpath: **int** ($arg as xs:anyAtomicType) as _xs:int_
xpath: **integer** ($arg as xs:anyAtomicType) as _xs:integer_
xpath: **iri-to-uri** ($iri as xs:string) as _xs:string_
xpath: **lang** ($testlang as xs:string [$node] as node()) as _xs:boolean_
xpath: **language** ($arg as xs:anyAtomicType) as _xs:language_
xpath: **last** () as _xs:integer_
xpath: **local-name** ([$arg] as node()) as _xs:string_
xpath: **local-name-from-QName** ($arg as xs:QName) as _xs:NCName_
xpath: **long** ($arg as xs:anyAtomicType) as _xs:long_
xpath: **lower-case** ($arg as xs:string) as _xs:string_
xpath: **matches** ($input as xs:string, $pattern as xs:string [$flags] as xs:string) as _xs:boolean_
xpath: **max** ($arg as xs:anyAtomicType* [$collation] as string) as _xs:anyAtomicType_
xpath: **min** ($arg as xs:anyAtomicType* [$collation] as string) as _xs:anyAtomicType_
xpath: **minutes-from-dateTime** ($arg as xs:dateTime) as _xs:integer_
xpath: **minutes-from-duration** ($arg as xs:duration) as _xs:integer_
xpath: **minutes-from-time** ($arg as xs:time) as _xs:integer_
xpath: **month-from-date** ($arg as xs:date) as _xs:integer_
xpath: **month-from-dateTime** ($arg as xs:dateTime) as _xs:integer_
xpath: **months-from-duration** ($arg as xs:duration) as _xs:integer_
xpath: **my:hatSize** ($arg as xs:anyAtomicType) as _my:hatSize_
xpath: **name** ([$arg] as node()) as _xs:string_
xpath: **namespace-uri** ([$arg] as node()) as _xs:anyURI_
xpath: **namespace-uri-for-prefix** ($prefix as xs:string, $element as element()) as _xs:anyURI_
xpath: **namespace-uri-from-QName** ($arg as xs:QName) as _xs:anyURI_
xpath: **negativeInteger** ($arg as xs:anyAtomicType) as _xs:negativeInteger_
xpath: **nilled** ($arg as node()) as _xs:boolean_
xpath: **node-name** ($arg as node()) as _xs:QName_
xpath: **nonNegativeInteger** ($arg as xs:anyAtomicType) as _xs:nonNegativeInteger_
xpath: **nonPositiveInteger** ($arg as xs:anyAtomicType) as _xs:nonPositiveInteger_
xpath: **normalize-space** ([$arg] as xs:string) as _xs:string_
xpath: **normalize-unicode** ($arg as xs:string [$normalizationForm] as xs:string) as _xs:string_
xpath: **normalizedString** ($arg as xs:anyAtomicType) as _xs:normalizedString_
xpath: **not** ($arg as item()*) as _xs:boolean_
xpath: **number** ([$arg] as xs:anyAtomicType) as _xs:double_
xpath: **one-or-more** ($arg as item()*) as _item()+_
xpath: **position** () as _xs:integer_
xpath: **positiveInteger** ($arg as xs:anyAtomicType) as _xs:positiveInteger_
xpath: **prefix-from-QName** ($arg as xs:QName) as _xs:NCName_
xpath: **remove** ($target as item()*, $position as xs:integer) as _item()*_
xpath: **replace** ($input as xs:string, $pattern as xs:string, $replacement as xs:string [$flags] as xs:string) as _xs:string_
xpath: **resolve-QName** ($qname as xs:string, $element as element()) as _xs:QName_
xpath: **resolve-uri** ($relative as xs:string [$base] as xs:string) as _xs:anyURI_
xpath: **reverse** ($arg as item()*) as _item()*_
xpath: **root** ([$arg] as node()) as _node()_
xpath: **round** ($arg as numeric) as _numeric_
xpath: **round-half-to-even** ($arg as numeric [$precision] as xs:integer) as _numeric_
xpath: **seconds-from-dateTime** ($arg as xs:dateTime) as _xs:decimal_
xpath: **seconds-from-duration** ($arg as xs:duration) as _xs:decimal_
xpath: **seconds-from-time** ($arg as xs:time) as _xs:decimal_
xpath: **short** ($arg as xs:anyAtomicType) as _xs:short_
xpath: **starts-with** ($arg1 as xs:string, $arg2 as xs:string [$collation] as xs:string) as _xs:boolean_
xpath: **static-base-uri** () as _xs:anyURI_
xpath: **string** ([$arg] as item() [$arg] as xs:anyAtomicType) as _xs:string_
xpath: **string-join** ($arg1 as xs:string*, $arg2 as xs:string) as _xs:string_
xpath: **string-length** ([$arg] as xs:string) as _xs:integer_
xpath: **string-to-codepoints** ($arg as xs:string) as _xs:integer*_
xpath: **subsequence** ($sourceSeq as item()*, $startingLoc as xs:double [$length] as xs:double) as _item()*_
xpath: **substring** ($sourceString as xs:string, $startingLoc as xs:double [$length] as xs:double) as _xs:string_
xpath: **substring-after** ($arg1 as xs:string, $arg2 as xs:string [$collation] as xs:string) as _xs:string_
xpath: **substring-before** ($arg1 as xs:string, $arg2 as xs:string [$collation] as xs:string) as _xs:string_
xpath: **sum** ($arg as xs:anyAtomicType* [$zero] as xs:anyAtomicType) as _xs:anyAtomicType_
xpath: **time** ($arg as xs:anyAtomicType) as _xs:time_
xpath: **timezone-from-date** ($arg as xs:date) as _xs:dayTimeDuration_
xpath: **timezone-from-dateTime** ($arg as xs:dateTime) as _xs:dayTimeDuration_
xpath: **timezone-from-time** ($arg as xs:time) as _xs:dayTimeDuration_
xpath: **token** ($arg as xs:anyAtomicType) as _xs:token_
xpath: **tokenize** ($input as xs:string, $pattern as xs:string [$flags] as xs:string) as _xs:string*_
xpath: **trace** ($value as item()*, $label as xs:string) as _item()*_
xpath: **translate** ($arg as xs:string, $mapString as xs:string, $transString as xs:string) as _xs:string_
xpath: **true** () as _xs:boolean_
xpath: **unordered** ($sourceSeq as item()*) as _item()*_
xpath: **unsignedByte** ($arg as xs:anyAtomicType) as _xs:unsignedByte_
xpath: **unsignedInt** ($arg as xs:anyAtomicType) as _xs:unsignedInt_
xpath: **unsignedLong** ($arg as xs:anyAtomicType) as _xs:unsignedLong_
xpath: **unsignedShort** ($arg as xs:anyAtomicType) as _xs:unsignedShort_
xpath: **untypedAtomic** ($arg as xs:anyAtomicType) as _xs:untypedAtomic_
xpath: **upper-case** ($arg as xs:string) as _xs:string_
xpath: **year-from-date** ($arg as xs:date) as _xs:integer_
xpath: **year-from-dateTime** ($arg as xs:dateTime) as _xs:integer_
xpath: **yearMonthDuration** ($arg as xs:anyAtomicType) as _xs:yearMonthDuration_
xpath: **years-from-duration** ($arg as xs:duration) as _xs:integer_
xpath: **zero-or-one** ($arg as item()*) as _item()_

Precedence Order

1	, (comma)	left-to-right
3	for, some, every, if	left-to-right
4	or	left-to-right
5	and	left-to-right
6	eq, ne, lt, le, gt, ge, =, !=, <, <=, >, >=, is, <<, >>	left-to-right
7	to	left-to-right
8	+, -	left-to-right
9	*, div, idiv, mod	left-to-right
10	union, |	left-to-right
11	intersect, except	left-to-right
12	instance of	left-to-right
13	treat	left-to-right
14	castable	left-to-right
15	cast	left-to-right
16	-(unary), +(unary)	right-to-left
17	?, *(OccurrenceIndicator), +(OccurrenceIndicator)	left-to-right
18	/, //	left-to-right
19	[ ], ( ), {}	left-to-right

Key

{Attribute Value Template}
Source (xslt or xpath), function name, ($parameter as type), as function return type. E.g. xpath: seconds-from-dateTime ($arg as xs:dateTime) as xs:decimal
optional arguments to functions are shown as [$parameter as type]

Location Paths [XPath §2]

Optional '/', zero or more **location steps**, separated by '/'

Location Steps [XPath §2.1]

Axis specifier, **node test**, zero or more **predicates**

Axis Specifiers [XPath §2.2]

ancestor::
ancestor-or-self::
attribute::
child::
descendant::
descendant-or-self::
following::
following-sibling::
namespace::
parent::
preceding::
preceding-sibling::
self::

Node Tests [XPath §2.3]

name
prefix:*name*
*
prefix:*
node()
text()
comment()
processing-instruction()
processing-instruction(*literal*)

Abbreviated Syntax for Location Paths

(nothing)	child::
@	attribute::
//	/descendant-or-self::node()/
.	self::node()
..	parent::node()
/	Node tree root

Predicate [XPath §2.4]

[*expr*]

Variable Reference [XPath §3.7]

$*qname*

Literal Result Elements [§7.1.1]

Any element not in the xsl: namespace and not an extension element

XSLT

http://www.w3.org/TR/xslt

XPath

http://www.w3.org/TR/xpath

XSL-List

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

Mulberry Technologies, Inc.

XPath Operators

Parentheses may be used for grouping.

Node-sets [XPath §3.3]

| [*expr*] / //

Booleans [XPath §3.4]

<=, <, >=, > **=, !=** **and** **or**

Numbers [XPath §3.5]

-expr ***, div, mod** **+, -**

XPath Core Function Library

Node Set Functions [XPath §4.1]

number **last**()
number **position**()
number **count**(*node-set*)
node-set **id**(*object*)
string **local-name**(*node-set*?)
string **namespace-uri**(*node-set*?)
string **name**(*node-set*?)

String Functions [XPath §4.2]

string **string**(*object*?)
string **concat**(*string*, *string*, *string**)
boolean **starts-with**(*string*, *string*)
boolean **contains**(*string*, *string*)
string **substring-before**(*string*, *string*)
string **substring-after**(*string*, *string*)
string **substring**(*string*, *number*, *number*?)
number **string-length**(*string*?)
string **normalize-space**(*string*?)
string **translate**(*string*, *string*, *string*)

Boolean Functions [XPath §4.3]

boolean **boolean**(*object*)
boolean **not**(*object*)
boolean **true**()
boolean **false**()
boolean **lang**(*string*)

Number Functions [XPath §4.4]

number **number**(*object*?)
number **sum**(*node-set*)
number **floor**(*number*)
number **ceiling**(*number*)
number **round**(*number*)

Mulberry Technologies, Inc.

XSLT and XPath Quick Reference

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.

XSLT Functions [§12, §15]

node-set **document**(*object*, *node-set*?)
node-set **key**(*string*, *object*)
string **format-number**(*number*, *string*, *string*?)
node-set **current**()
string **unparsed-entity-uri**(*string*)
string **generate-id**(*node-set*?)
object **system-property**(*string*)
boolean **element-available**(*string*)
boolean **function-available**(*string*)

Node Types [XPath §5]

Root
Element
Attribute
Namespace
Processing Instruction
Comment
Text

Object Types [§11.1, XPath §1]

boolean	True or false
number	Floating-point number
string	UCS characters
node-set	Set of nodes selected by a path
Result tree fragment	XSLT only. Fragment of the result tree

Expression Context [§4, XPath §1]

Context node (a node)
Context position (a number)
Context size (a number)
Variable bindings in scope
Namespace declarations in scope
Function library

Built-in Template Rules [§5.8]

```
<xsl:template match="*|/">
    <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="*|/" mode="m">
    <xsl:apply-templates mode="m"/>
</xsl:template>
```

```
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>
```

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

Built-in template rule for namespaces is to do nothing

Mulberry Technologies, Inc.

XSLT Elements

Stylesheet Element [§2.2]

```
<xsl:stylesheet version="1.0" id="id"
    extension-element-prefixes="tokens"
    exclude-result-prefixes="tokens"
    xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform"> xsl:import*, top-level elements
</xsl:stylesheet>
```

xsl:transform is a synonym for xsl:stylesheet

Combining Stylesheets [§2.6]

```
<xsl:include href="uri-reference"/>

<xsl:import href="uri-reference"/>
```

Whitespace Stripping [§3.4]

```
<xsl:strip-space elements="tokens"/>

<xsl:preserve-space elements="tokens"/>
```

Defining Template Rules [§5.3]

```
<xsl:template match="pattern" name="qname"
    priority="number" mode="qname">
    xsl:param* followed by text, literal result elements
    and/or XSL elements </xsl:template>
```

Applying Template Rules [§5.4]

```
<xsl:apply-templates select="node-set-exp"
    mode="qname"/>
<xsl:apply-templates select="node-set-exp"
    mode="qname">
    (xsl:sort | xsl:with-param)* </xsl:apply-templates>
```

Overriding Template Rules [§5.6]

```
<xsl:apply-imports/>
```

Named Templates [§6]

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

Namespace Alias [§7.1.1]

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

Creating Elements [§7.1.2]

```
<xsl:element name="{qname}"
    namespace="{uri-reference}"
    use-attribute-sets="qnames">…</xsl:element>
```

Creating Attributes [§7.1.3]

```
<xsl:attribute name="{qname}"
    namespace="{uri-reference}">…</xsl:attribute>
```

Named Attribute Sets [§7.1.4]

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

Creating Text [§7.2]

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

Processing Instructions [§7.3]

```
<xsl:processing-instruction name="{ncname}">
    …</xsl:processing-instruction>
```

Creating Comments [§7.4]

```
<xsl:comment>…</xsl:comment>
```

Copying [§7.5]

```
<xsl:copy use-attribute-sets="qnames">
    …</xsl:copy>
```

Generating Text [§7.6.1]

```
<xsl:value-of select="string-expr"
    disable-output-escaping="yes|no"/>
```

Attribute Value Templates [§7.6.2]

```
<element attribute="{expr}"/>
```

Numbering [§7.7]

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

Repetition [§8]

```
<xsl:for-each select="node-set-expr">
    xsl:sort*, …</xsl:for-each>
```

Conditional Processing [§9]

```
<xsl:if test="boolean-expr">…</xsl:if>

<xsl:choose>
    <xsl:when test="expr">…</xsl:when>+
    <xsl:otherwise>…</xsl:otherwise>?
</xsl:choose>
```

Sorting [§10]

```
<xsl:sort select="string-expr" lang="{nmtoken}"
    data-type="{text|number|qname-but-not-
    ncname}" order="{ascending|descending}"
    case-order="{upper-first|lower-first}"/>
```

Variables and Parameters [§11]

```
<xsl:variable name="qname" select="expr"/>
<xsl:variable name="qname">…</xsl:variable>

<xsl:param name="qname" select="expr"/>
<xsl:param name="qname">…</xsl:param>
```

Using Values [§11.3]

```
<xsl:copy-of select="expr"/>
```

Passing Parameters [§11.6]

```
<xsl:with-param name="expr" select="expr"/>
<xsl:with-param name="expr">…</xsl:with-param>
```

Keys [§12.2]

```
<xsl:key name="qname" match="pattern"
    use="expr"/>
```

Number Formatting [§12.3]

```
<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"/>
```

Messages [§13]

```
<xsl:message terminate="yes|no">
    …</xsl:message>
```

Fallback [§15]

```
<xsl:fallback>…</xsl:fallback>
```

Output [§16]

```
<xsl:output
    method="xml|html|text|qname-but-not-ncname"
    version="nmtoken" encoding="string"
    omit-xml-declaration="yes|no"
    doctype-public="string" doctype-system="string"
    standalone="yes|no" indent="yes|no"
    cdata-section-elements="qnames"
    media-type="string"/>
```

Key

xsl:stylesheet	Element
version=	Required attribute
version=	Optional attribute
{expr}	Attribute value template. Text between any { and } is evaluated as an expression. Attribute value must evaluate to indicated attribute type.
…	Anything allowed in a template
|	Separates alternative values
?	Zero or one occurrences
*	Zero or more occurrences
+	One or more occurrences
#PCDATA	Character data

Attribute Value Types

1.0	Literal value
boolean-expr	Expression returning boolean value
char	Single character
expr	Expression
id	XML name used as identifier
ncname	XML name not containing a colon (:)
node-set-expr	Expression returning a node set
number-expr	Expression returning a number
pattern	XSLT pattern
prefix	Namespace prefix
qname	Namespace-qualified XML name comprising local part and optional prefix
qname-but-not-ncname	Namespace-qualified name comprising local part and prefix
token	Meaning varies with context. See Rec.
uri-reference	Reference to Universal Resource Identifier

XQuery v1.0 and XPath v2.0 Functions and Operators Quick Reference

ver 1/0

©2008 D Vint Productions
xmlhelp@dvint.com
http://www.xml.dvint.com

1 Namespaces §1

- http://www.w3.org/2001/XMLSchema for constructors -- associated with xs
- http://www.w3.org/2005/xpath-functions for functions -- associated with fn
- http://www.w3.org/2005/xqt-errors -- associated with err

Functions defined with the op prefix are not available directly to users, and there is no requirement that implementations should actually provide these functions.No namespace is associated with the op prefix.

numeric is used in function signatures as a shorthand to indicate the four numeric types: xs:integer, xs:decimal, xs:float and xs:double

Some functions accept a single value or the empty sequence as an argument and some may return a single value or the empty sequence. This is indicated in the function signature by following the parameter or return type name with a question mark: "?".

2 Accessors §2

- fn:node-name($node?) Returns an expanded-QName for node kinds that can have names.
- fn:nilled($node?) Returns an xs:boolean indicating whether the argument node is "nilled".
- fn:string() Returns xs:string evaluates the context item
- fn:string($item?) Returns xs:string
- fn:data($item*) takes a sequence of items and returns a sequence of atomic values.
- fn:base-uri() Returns xs:anyURI? evaluates the context item
- fn:base-uri($node?) Returns the value of the base-uri
- fn:document-uri($node?) Returns the value of the document-uri property for $arg .

3 The Error Function §3

- fn:error() Returns none
- fn:error($error) Returns none
- fn:error($error , $description) Returns none
- fn:error($error, $description,$error-object*) Returns none

While this function never returns a value, an error is returned to the external processing environment as an xs:anyURI or an xs:QName. An error xs:QName with namespace URI NS and local part LP will be returned as the xs:anyURI NS#LP.

- fn:error() Returns http://www.w3.org/2005/xqt-errors#FOER0000
- fn:error(fn:QName('http://www.example.com/HR', 'myerr:toohighsal'), 'Does not apply because salary is too high') Returns http://www.example.com/HR#toohighsal and the xs:string "Does not apply because salary is too high"

4 The Trace Function §4

- fn:trace($item*, $label) Returns item()* Provides an execution trace intended to be used in debugging queries.
- fn:trace($v, 'the value of $v is:')

5 Constructor Functions §5

Every built-in atomic type that is defined in XML Schema Part 2: Datatypes, except xs:anyAtomicType and xs:NOTATION, has an associated constructor function. And there is a special function for dateTime:

- fn:dateTime($date?, $time?) Returns xs:dateTime?

For every atomic type in the static context that is derived from a primitive type, there is a constructor function (whose name is the same as the name of the type) whose effect is to create a value of that type from the supplied argument.

- my:hatSize($arg?) as my:hatSize?
- 17 cast as apple
- declare default function namespace ""; apple(17)

6 Functions and Operators on Numerics §6

- fn:abs($numeric?) Returns the absolute value of the argument.
- fn:ceiling($numeric?) Returns the smallest number with no fractional part that is greater than or equal to the argument.
- fn:floor($numeric?) Returns the largest number with no fractional part that is less than or equal to the argument.
- fn:round($numeric?) Rounds to the nearest number with no fractional part.
- fn:round-half-to-even($numeric?) Returns numeric?
- fn:round-half-to-even($numeric?, $precision) Returns numeric?Takes a number and a precision and returns a number rounded to the given precision. If the fractional part is exactly half, the result is the number whose least significant digit is even.
  - fn:round-half-to-even(0.5) returns 0.
  - fn:round-half-to-even(1.5) returns 2.
  - fn:round-half-to-even(2.5) returns 2.
  - fn:round-half-to-even(3.567812E+3, 2) returns 3567.81E0.
  - fn:round-half-to-even(4.7564E-3, 2) returns 0.0E0.
  - fn:round-half-to-even(35612.25, -2) returns 35600.

7 Functions on Strings §7

The first character of a string is located at position 1, not position 0.

- fn:codepoints-to-string(xs:integer*) Returns a xs:string from a sequence of code points.
- fn:codepoints-to-string((2309, 2358, 2378, 2325)) returns "अशोक"
- fn:string-to-codepoints(xs:string?) Returns the sequence of code points that constitute an xs:string
  - fn:string-to-codepoints("ThÈrËse") Returns the sequence (84, 104, 233, 114, 232, 115, 101)
- fn:compare($comparand1 as xs:string?, $comparand2?) Returns xs:integer?
- fn:compare($comparand1?, $comparand2?, $collation) Returns -1, 0, or 1
  - fn:compare('abc', 'abc') Returns 0.
  - fn:compare('Strasse', 'Straße') Returns 0 if and only if the default collation includes provisions that equate "ss" and the (German) character "?" ("sharp-s").
  - fn:compare('Strasse', 'Straße', 'deutsch') Returns 0 if the collation identified by the relative URI value "deutsch" includes provisions that equate "ss" and the (German) character "?" ("sharp-s").
- fn:codepoint-equal($comparand1, $comparand2) Returns true or false depending on whether the value of $comparand1 is equal to the value of $comparand2, according to the Unicode code point collation.
- fn:compare($comparand1, $comparand2) Returns xs:integer?
- fn:compare($comparand1, $comparand2, $collation) Returns xs:integer?
- fn:codepoint-equal($comparand1, $comparand2) Returns xs:boolean?
- fn:concat(xs:anyAtomicType?, xs:anyAtomicType?, ...) Returns xs:string
- fn:string-join($string*, $string) Returns a xs:string created by concatenating the members of the $arg1 sequence using $arg2 as a separator.
  - fn:string-join(('Now', 'is', 'the', 'time', '...'), ' ') Returns " Now is the time ... "
  - fn:string-join(('Blow, ', 'blow, ', 'thou ', 'winter ', 'wind!'), '') Returns " Blow, blow, thou winter wind! "
  - fn:string-join((), 'separator') Returns ""
- fn:substring($sourceString, $startingLoc) Returns xs:string
- fn:substring($sourceString, $startingLoc, $length) Returns xs:string
- fn:substring-before($string?, $pattern?) Returns xs:string
- fn:substring-before($string?,$pattern?,$collation) Returns xs:string
- fn:substring-after($string?, $pattern?) Returns xs:string
- fn:substring-after($string?, $pattern?, $collation) Returns xs:string
- fn:string-length() Returns xs:integer
- fn:string-length($string?) Returns xs:integer
- fn:normalize-space() Returns xs:string Strips leading and traling whitespace and replaces sequences of whitespace with one
- fn:normalize-space(xs:string?) Returns xs:string
- fn:normalize-unicode($string?) Returns xs:string
- fn:normalize-unicode($string?, $normalizationForm) Returns xs:string Returns the value of $arg normalized according to the normalization criteria for a normalization form identified by the value of $normalizationForm. $normalizationForm can be: "NFC","NFD", "NFKC", "NFKD","FULLY-NORMALIZED", or the zero-length string.
- fn:upper-case($string?) Returns xs:string
- fn:lower-case($string?) Returns xs:string
- fn:translate($string?, $mapString, $transString) Returns xs:string
  - fn:translate("bar","abc","ABC") Returns "BAr"
  - fn:translate("--aaa--","abc-","ABC") Returns "AAA".
  - fn:translate("abcdabc", "abc", "AB") Returns "ABdAB".
- fn:encode-for-uri($uri-part) Returns xs:string
  - fn:encode-for-uri("http://www.example.com/00/Weather/CA/Los%20Angeles#ocean") Returns "http%3A%2F%2Fwww.example.com%2F00%2FWeather%2FCA%2FLos%2520Angeles%23ocean".
  - concat("http://www.example.com/", encode-for-uri("~bÈbÈ")) Returns "http://www.example.com/~b%C3%A9b%C3%A9".
  - concat("http://www.example.com/", encode-for-uri("100% organic")) Returns "http://www.example.com/100%25%20organic".
- fn:iri-to-uri($iri) Returns xs:string
  - fn:iri-to-uri("http://www.example.com/00/Weather/CA/Los%20Angeles#ocean") Returns "http://www.example.com/00/Weather/CA/Los%20Angeles#ocean".
  - fn:iri-to-uri("http://www.example.com/~bÈbÈ") returns "http://www.example.com/~b%C3%A9b%C3%A9".
- fn:escape-html-uri($uri) Returns xs:string
  - fn:escape-html-uri("http://www.example.com/00/Weather/CA/Los Angeles#ocean") Returns "http://www.example.com/00/Weather/CA/Los Angeles#ocean".
  - fn:escape-html-uri("javascript:if (navigator.browserLanguage == 'fr') window.open('http://www.example.com/~bÈbÈ');") Returns "javascript:if (navigator.browserLanguage == 'fr') window.open('http://www.example.com/~b%C3%A9b%C3%A9');".
- fn:contains($string?, $pattern?) Returns xs:boolean
- fn:contains($string?, $pattern?, $collation) Returns xs:boolean
- fn:starts-with($string?, $pattern?) Returns xs:boolean
- fn:starts-with($string?, $pattern?, $collation) Returns xs:boolean
- fn:ends-with($string?, $pattern?) Returns xs:boolean
- fn:ends-with($string?, $pattern?, $collation) Returns xs:boolean
- fn:matches($input, $pattern) Returns xs:boolean
- fn:matches($input, $pattern, $flags) Returns xs:boolean
- fn:replace($input, $pattern, $replacement) Returns xs:string
- fn:replace($input, $pattern, $replacement, $flags) Returns xs:string
- fn:tokenize($input, $separator) Returns xs:string*
- fn:tokenize($input, $separator, $flags) Returns xs:string*

8 Functions on anyURI §8

- fn:resolve-uri($relative) Returns xs:anyURI?
- fn:resolve-uri($relative, $base) Returns xs:anyURI?

9 Functions and Operators on Boolean Values §9

- fn:true() Returns xs:boolean
- fn:false() Returns xs:boolean
- fn:not(item()*) Returns xs:boolean

10 Functions and Operators on Durations, Dates and Times §10

- fn:years-from-duration($duration?) Returns xs:integer?
- fn:months-from-duration($duration?) Returns xs:integer?
- fn:days-from-duration($duration?) Returns xs:integer?

- `fn:hours-from-duration($duration?)` Returns `xs:integer?`
- `fn:minutes-from-duration($duration?)` Returns `xs:integer?`
- `fn:seconds-from-duration($duration?)` Returns `xs:decimal?`
- `fn:year-from-dateTime($dateTime?)` Returns `xs:integer?`
- `fn:month-from-dateTime($dateTime?)` Returns `xs:integer?`
- `fn:day-from-dateTime($dateTime?)` Returns `xs:integer?`
- `fn:hours-from-dateTime($dateTime?)` Returns `xs:integer?`
- `fn:minutes-from-dateTime($dateTime?)` Returns `xs:integer?`
- `fn:seconds-from-dateTime($dateTime?)` Returns `xs:decimal?`
- `fn:timezone-from-dateTime($dateTime?)` Returns `xs:dayTimeDuration?`
- `fn:year-from-date($date?)` Returns `xs:integer?`
- `fn:month-from-date($date?)` Returns `xs:integer?`
- `fn:day-from-date($date?)` Returns `xs:integer?`
- `fn:timezone-from-date($date?)` Returns `xs:dayTimeDuration?`
- `fn:hours-from-time($time?)` Returns `xs:integer?`
- `fn:minutes-from-time($time?)` Returns `xs:integer?`
- `fn:seconds-from-time($time?)` Returns `xs:decimal?`
- `fn:timezone-from-time($time?)` Returns `xs:dayTimeDuration?`
- `fn:adjust-dateTime-to-timezone($dateTime?)` Returns `xs:dateTime?`
- `fn:adjust-dateTime-to-timezone($dateTime?, $timezone)` Returns `xs:dateTime?`
- `fn:adjust-date-to-timezone($date?)` Returns `xs:date?`
- `fn:adjust-date-to-timezone($date?, $timezone?)` Returns `xs:date?`
- `fn:adjust-time-to-timezone($time?)` Returns `xs:time?`
- `fn:adjust-time-to-timezone($time?, $timezone?)` Returns `xs:time?`

11 Functions Related to QNames §11

- `fn:resolve-QName($qname, $element)` Returns expanded `xs:QName?`
- `fn:QName($URI, $QName)` Returns an `xs:QName` with the namespace URI given in `$URI`
- `fn:prefix-from-QName($paramQName)` Returns `xs:NCName?`
- `fn:local-name-from-QName($paramQName)` Returns the local name
- `fn:namespace-uri-from-QName($paramQName)` Returns the namespace URI for the `xs:QName` argument. If the `xs:QName` is in no namespace, the zero-length string is returned
- `fn:namespace-uri-for-prefix($prefix, $element)` Returns the namespace URI of one of the in-scope namespaces for the given element, identified by its namespace prefix
- `fn:in-scope-prefixes($element)` Returns the prefixes of the in-scope namespaces for the given element

12 Functions and Operators on Nodes §14

- `fn:name()` Returns `xs:string`
- `fn:name($node?)` Returns `xs:string`
- `fn:local-name()` Returns `xs:string`
- `fn:local-name($node?)` Returns `xs:string`
- `fn:namespace-uri()` Returns `xs:anyURI`
- `fn:namespace-uri($node?)` Returns `xs:anyURI`
- `fn:number()` Returns `xs:double`
- `fn:number($arg?)` Returns `xs:double`
- `fn:lang($testlang)` Returns `xs:boolean`
- `fn:lang($testlang, $node)` Returns `xs:boolean`
- `fn:root()` Returns `node()`
- `fn:root($node)` Returns the root of the tree to which the node argument belongs

13 Functions and Operators on Sequences §15

- `fn:boolean($item*)` Returns `xs:boolean`
- `fn:index-of($seqParam*, $srchParam)` Returns `xs:integer*`
- `fn:index-of($seqParam*, $srchParam, $collation)` Returns `xs:integer*`
- `fn:empty($item*)` Returns `xs:boolean`
- `fn:exists($item*)` Returns `xs:boolean`
- `fn:distinct-values($arg*)` Returns `xs:anyAtomicType*`
- `fn:distinct-values($arg*, $collation)` Returns `xs:anyAtomicType*`
- `fn:insert-before($targetitem*, $position, $insertsitem*)` Returns `item()*`
- `fn:remove($targetitem*, $position)` Returns `item()*`
- `fn:reverse($item*)` Returns `item()*`
- `fn:subsequence($sourceSeq*, $startingLoc)` Returns `item()*`
- `fn:subsequence($sourceSeq*, $startingLoc, $length)` Returns `item()*`
  - `fn:unordered($sourceSeq*)` Returns `item()*`
- `fn:zero-or-one($item*)` Returns the input sequence if it contains zero or one items
- `fn:one-or-more($item*)` Returns the input sequence if it contains one or more items
- `fn:exactly-one($item*)` Returns the input sequence if it contains exactly one item
- `fn:deep-equal($arg1item*, $arg2item*)` Returns `true` if the two arguments have items that compare equal in corresponding positions
- `fn:deep-equal($arg1item*, $arg2item*, $collation)` Returns `xs:boolean`
- `fn:count(item()*)` Returns `xs:integer`
- `fn:avg($arg*)` Returns `xs:anyAtomicType?`
- `fn:max($arg*)` Returns `xs:anyAtomicType?`
- `fn:max($arg*, $collation)` Returns `xs:anyAtomicType?`
- `fn:min($arg*)` Returns `xs:anyAtomicType?`
- `fn:min($arg*, $collation)` Returns `xs:anyAtomicType?`
- `fn:sum($arg*)` Returns `xs:anyAtomicType`
- `fn:sum($arg*, $emptySeqreturnvalue?)` Returns `xs:anyAtomicType?`
- `fn:id($string*)` Returns the sequence of element nodes having an ID value matching the one or more of the supplied `IDREF` values
- `fn:id($string*, $node))` Returns `element()*`
- `fn:idref($string*)` Returns the sequence of element or attribute nodes with an IDREF value matching one or more of the supplied ID values.
- `fn:idref($string*, $node)` Returns `node()*`
- `fn:doc($uri?)` Retrieves a document using an `xs:anyURI`, which may include a fragment identifier
- `fn:doc-available($uri)` Returns `xs:boolean`
- `fn:collection()` This function takes an xs:string as argument and returns a sequence of nodes obtained by interpreting $arg as an xs:anyURI and resolving it according to the mapping specified in Available collections. If Available collections provides a mapping from this string to a sequence of nodes, the function returns that sequence
- `fn:collection($string?)` Returns `node()*`

14 Context Functions §16

- `fn:position()` Returns `xs:integer`
- `fn:last()` Returns `xs:integer`
- `fn:current-dateTime()` Returns `xs:dateTime`
- `fn:current-date()` Returns `xs:date`
- `fn:current-time()` Returns `xs:time`
- `fn:implicit-timezone()` Returns `xs:dayTimeDuration`
- `fn:default-collation()` Returns `xs:string`
- `fn:static-base-uri()` Returns `xs:anyURI?`

15 Regular Expression Syntax §7.6.1

This section describes extensions to the XML Schema regular expressions syntax that reinstate capabilities that were left out of the Schema syntax.

- Two meta-characters, `^` and `$` are added. By default, the meta-character `^` matches the start of the entire string, while `$` matches the end of the entire string. In multi-line mode, `^` matches the start of any line (that is, the start of the entire string, and the position immediately after a newline character), while `$` matches the end of any line.
- *Reluctant quantifiers* are supported. They are indicated by a " `?` " following a quantifier. Specifically:
  - `X??` matches X, once or not at all
  - `X*?` matches X, zero or more times
  - `X+?` matches X, one or more times
  - `X{n}?` matches X, exactly n times
  - `X{n,}?` matches X, at least n times
  - `X{n,m}?` matches X, at least n times, but not more than m times
- Sub-expressions (groups) within the regular expression are recognized. The sub-expressions are numbered according to the position of the opening parenthesis in left-to-right order within the top-level regular expression: the first opening parenthesis identifies captured substring 1, the second identifies captured substring 2, and so on. 0 identifies the substring captured by the entire regular expression. If a sub-expression matches more than one substring (because it is within a construct that allows repetition), then only the *last* substring that it matched will be captured.
- Back-references are allowed.

Flags §7.6.1.1

All these functions provide an optional parameter, `$flags`, to set options for the interpretation of the regular expression. The following options are defined:

- `s`: If present, the match operates in "dot-all" mode. (Perl calls this the single-line mode.) If the `s` flag is not specified, the meta-character `.` matches any character except a newline (`#x0A`) character. In dot-all mode, the meta-character `.` matches any character whatsoever.
- `m`: If present, the match operates in multi-line mode.
- `i`: If present, the match operates in case-insensitive mode.
- `x`: If present, whitespace characters (#x9, #xA, #xD and #x20) in the regular expression are removed prior to matching. This flag can be used, for example, to break up long regular expressions into readable lines. `fn:matches("helloworld", "hello world", "x")` returns `true`

16 Regular Expressions from Schema Speciification

Special Characters needing to be escaped with a '\'

- `\ | . - ^ ? * + { } ( ) [ ]`

Character References

&#x4E; or &#99; for hex or decimal XML character references

Interval Operators

- {x,y} range x to y, {x,} at least x, {x} exactly x, i.e. {4,8} 4 to 8
- Repetitions * + ?

Character Range Expressions

- [a-zA-Z] = character a to z upper and lower case
  [0-9] = digits 0 to 9

Special Character Sequences

\n	newline
\r	return
\t	tab
. (dot)	all characters except newline and return
\s	space characters (space, tab, newline, return)
\S	non-Space characters
\i	initial XML name characters (letter _ ;)
\I	not initial XML name characters
\c	XML NameChar characters
\C	not XML NameChar characters
\d	decimal digits
\D	not decimal digits
\w	XML Letter or Digit characters
\W	not XML Letter or Digit characters

\p{IsBasicLatin}	block escape identifying ASCII characters, similar IsGreek, IsHebrew, IsThai for these ranges of Unicode blocks
\p{L}	all Letters
\p{M}	all Marks
\p{N}	all Numbers
\p{P}	all Punctuation
\p{Z}	all Separators
\p{S}	all Symbols

\p{C}	all Others. Additional modifying values like Lu = uppercase,
	Ll = lowercase, Nd = decimal digit,
	Sm = math symbols, Sc = currency

\P{}	not the block or category, \P{IsGreek} = not Greek block

Pattern Examples

Chapter \d	Chapter 0, Chapter 1, Chapter 2....
Chapter\s\w	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a word character (XML 1.0 Letter or Digit)
Espan&#xF1;ola	Española
\p{Lu}	any uppercase character, the value of \p{} (e.g. "Lu") is defined by Unicode
a*x	x, ax, aax, aaax....
a?x	ax, x
a+x	ax, aax, aaax....
(a|b)+x	ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax....
[^0-9]x	any non-digit character followed by the character x
\Dx	any non-digit character followed by the character x
.x	any character followed by the character x
.*abc.*	1x2abc, abc1x2, z3456abchooray....
ab{2,4}x	abbx, abbbx, abbbbx

xmlhelp@dvint.com
http://www.xml.dvint.com

XPath v2.0 Quick Reference

ver 1/0

©2008 D Vint Productions
xmlhelp@dvint.com
http://www.xml.dvint.com

1 Namespaces

`http://www.w3.org/2001/XMLSchema`, prefixed as `xs`.
`http://www.w3.org/2005/xqt-errors` prefixed as `err`
`http://www.w3.org/2005/xpath-functions` prefixed as `fn`

2 Document Order §2.4.1

Document order is the order in which nodes appear in the XML serialization of a document. Document order is **stable**, which means that the relative order of two nodes will not change during the processing of a given expression, even if this order is implementation-dependent. The node ordering that is the reverse of document order is called **reverse document order**.

3 Atomization §2.4.2

Atomization is applied to a value when the value is used in a context in which a sequence of atomic values is required. The result of atomization is either a sequence of atomic values or a type error. **Atomization** of a sequence is defined as the result of invoking the `fn:data` function on the sequence.

Atomization is used in processing the following types of expressions:

- Arithmetic expressions
- Comparison expressions
- Function calls and returns
- Cast expressions

4 Effective Boolean Value §2.4.3

The **effective boolean value** of a value is defined as the result of applying the `fn:boolean` function to the value. The effective boolean value of a sequence is computed implicitly during processing of the following types of expressions:

- Logical expressions (`and`, `or`)
- The `fn:not` function
- Certain types of
  - predicates, such as `a[b]`
  - Conditional expressions (`if`)
  - Quantified expressions (`some`, `every`)
  - General comparisons, in XPath 1.0 compatibility mode.

5 Types §2.5

A **sequence type** is a type that can be expressed using the SequenceType syntax. Sequence types are used whenever it is necessary to refer to a type in an XPath expression.

A **schema type** is a type that is or could be defined using the facilities of XML Schema. Every schema type is either a **complex type** or a **simple type**; simple types are further subdivided into **list types**, **union types**, and **atomic types**.

Atomic types represent the intersection between the categories of sequence type and schema type. An atomic type, such as `xs:integer` or `my:hatsize`, is both a sequence type and a schema type.

Predefined Schema Types §2.5.1

- `xs:untyped` is used for an element node that has not been validated, or has been validated in `skip` mode.
- `xs:untypedAtomic` is an atomic type that is used to denote untyped atomic data, such as text that has not been assigned a more specific type.
- `xs:dayTimeDuration` is derived by restriction from `xs:duration` restricted to contain only day, hour, minute, and second components.
- `xs:yearMonthDuration` is derived by restriction from `xs:duration` restricted to contain only year and month components.
- `xs:anyAtomicType` is an atomic type that includes all atomic values (and no values that are not atomic). Its base type is `xs:anySimpleType` from which all simple types, including atomic, list, and union types, and primitive atomic types, such as `xs:integer`, `xs:string`.

Sequence Types §2.5.3

`empty-sequence()` or ItemType Occurrence_Indicator
ItemType = KindTest or `item()` or AtomicType
AtomicType = QName

An Occurrence Indicator specifies the number of items in a sequence, as follows:

- `?` matches zero or one items
- `*` matches zero or more items
- `+` matches one or more items
- none matches one item only and is required

As a consequence of the following rules, any sequence type whose occurrence indicator is `*` or `?` matches a value that is an empty sequence.

- `empty-sequence()` matches a value that is the empty sequence.
- An itemType with an occurrence indicator matches a value if the number of items in the value matches the occurrence indicator and the ItemType matches each of the items in the value.

6 Comments §2.6

Comments are strings, delimited by the symbols `(:` and `:)`. Comments are lexical constructs only, and do not affect expression processing. Comments may be nested and used anywhere ignorable whitespace is allowed .

7 Primary Expressions §3.1

Literals §3.1.1

```
Integer = 123 Decimals = 1.23 Doubles = 1.23 e+2
“String” or ‘string’    Escape Quote = '""'     Escape Apos = "''"
```

Variable References §3.1.2

`$QName` Two variable references are equivalent if their local names are the same and their namespace prefixes are bound to the same namespace URI in the statically known namespaces. An unprefixed variable reference is in no namespace.

Parenthesized Expressions §3.1.3

empty sequence = ()

Parentheses enforce a particular evaluation order in expressions that contain multiple operators.

Context Item Expression §3.1.4

- A **context item expression** evaluates to the context item, which may be either a node (as in the expression `fn:doc("bib.xml")/books/book[fn:count(./author)>1]`) or an atomic value (as in the expression `(1 to 100)[. mod 5 eq 0]`).
- The **context item** is the item currently being processed. An item is either an atomic value or a node.When the context item is a node, it can also be referred to as the context node. The context item is returned by an expression consisting of a single dot (`.`).
- If the context item is undefined, a context item expression raises a dynamic error

3.1.5 Function Calls

- A function call consists of a QName followed by a parenthesized list of zero or more expressions, called arguments. If the QName in the function call has no namespace prefix, it is considered to be in the default function namespace.
- If the expanded QName and number of arguments in a function call do not match the name and arity of a function signature in the static context, a static error is raised.

8 Path Expressions §3.2

A series of one or more steps, separated by "/" or "//", and optionally beginning with "/" or "//".

Steps §3.2.1

Axis specifier, node test, zero or more predicates

Axes §3.2.1.1

- **Forward**
  - `child::` `descendant::` `descendant-or-self::` `self::` `following::` `following-sibling::`
- **Reverse**
  - `ancestor::` `ancestor-or-self::` `parent::` `preceding::` `preceding-sibling::`
- **Other**
  - `namespace::` `attribute::`

Predicates §3.2.2

- `[expr]`

Abbreviated Syntax §3.2.4

- (nothing) = `child::`
- `@` = `attribute::`
- `//` = /descendant-or-self::node()/
- `.` = `self::node()`
- `..` = `parent::node()`
- `/` = Node tree root

Node/Kind Tests §2.5.4.1, 3.2.1.2

- `name`
- `prefix:name`
- `*`
- `prefix:*`
- `attribute()` `attribute(*)` `attribute(*, TypeName)` `attribute(AttributeName)` `attribute(AttributeName, TypeName)`
- `comment()`
- `document-node()` `document-node(element(book))`
- `element()` `element(*)` `element(*, TypeName ?)` `element(*, TypeName)` `element(ElementName)` `element(ElementName, TypeName ?)` `element(ElementName, TypeName)`
- `item()`
- `node()`
- `processing-instruction()` `processing-instruction(N)`
- `schema-attribute(AttributeName)` `schema-element(ElementName)`
- `text()`

9 Sequence Expressions §3.3

Sequences are never nested--for example, combining the values 1, (2, 3), and ( ) into a single sequence results in the sequence (1, 2, 3).

Constructing Sequences §3.3.1

The **comma operator**, evaluates each of its operands and concatenates the resulting sequences, in order, into a single result sequence.

- `10, 1, 2, 3, 4)` a sequence of five integers:
- `(10, (1, 2), (), (3, 4))` four sequences evaluates to `10, 1, 2, 3, 4`.

A **range expression** result is a sequence containing the two integer operands and every integer between the two operands, in increasing order.

- `(10, 1 to 4)` evaluates to the sequence `10, 1, 2, 3, 4`.
- `15 to 10` a sequence of length zero.
- `fn:reverse(10 to 15)` evaluates to the sequence `15, 14, 13, 12, 11, 10`.

Filter Expressions §3.3.2

- `$products[price gt 100]` = return only those products whose price is greater than 100
- `(1 to 100)[. mod 5 eq 0]` the integers from 1 to 100 that are divisible by 5
- `(21 to 29)[5]` result is the integer 25
- `$orders[fn:position() = (5 to 9)]` returns the fifth through ninth items in the sequence bound to variable `$orders`
- `$book/(chapter | appendix)[fn:last()]` returns the last chapter or appendix within the book bound to variable `$book`
- `fn:doc("zoo.xml")/fn:id('tiger')` returns the element node within the specified document whose ID value is `tiger`

Combining Node Sequences §3.3.3

`union | intersect except`

All these operators eliminate duplicate nodes from their result sequences based on node identity. The resulting sequence is returned in document order. If an operand contains an item that is not a node, a type error is raised.

- `$seq1` is bound to (A, B) `$seq2` is bound to (A, B) `$seq3` is bound to (B, C)
- `$seq1 union $seq2` evaluates to the sequence (A, B)
- `$seq1 intersect $seq2` evaluates to the sequence (A, B)
- `$seq2 except $seq3` evaluates to the sequence containing A only

10 Arithmetic Expressions §3.4

`-expr +expr * div idiv mod + -`

`idiv` divides the first argument by the second, and returns the integer obtained by truncating the fractional part of the result.

`mod` returns the remainder resulting from dividing $arg1, the dividend, by $arg2, the divisor.

11 Comparison Expressions §3.5

Comparison expressions allow two values to be compared. The kinds of comparison expressions are **value**, **general**, and **node** comparisons.

`eq ne lt le gt ge = != < <= > >= is <<(preceeds) >>(follows)`

Note: When an XPath expression is written within an XML document, the XML escaping rules for special characters must be followed; thus "<" must be written as "`&lt;`".

Value Comparisons §3.5.1

`eq ne lt le gt ge`

Value comparisons are used for comparing single values. If the result of atomization is an empty sequence, the result of the comparison is an empty sequence. If the result of atomization is a sequence containing more than one value, a type error is raised.

- `$book1/author eq "Kennedy"` true only if the result of atomization is the value "Kennedy" as an instance of `xs:string` or `xs:untypedAtomic`.
- `//product[weight gt 100]` selects products whose weight is greater than 100. For any product that does not have a weight subelement, the value of the predicate is the empty sequence, and the product is not selected.
- `my:hatsize(5) eq my:shoesize(5)` true if my:hatsize and my:shoesize are both user-defined types that are derived by restriction from a numeric type.
- `fn:QName("http://example.com/ns1", "this:color") eq fn:QName("http://example.com/ns1", "that:color")`

General Comparisons §3.5.2

`= != < <= > >=`

General comparisons are quantified comparisons that may be applied to operand sequences of any length. The result of a general comparison that does not raise an error is always true or false.

- `$book1/author = "Kennedy"` true if the typed value of any author subelement of $book1 is "Kennedy" as an instance of xs:string or xs:untypedAtomic:
- `(1, 2) = (2, 3)` is true
- `(2, 3) = (3, 4)` is true
- `(1, 2) = (3, 4)` is false
- `(1, 2) = (2, 3)` is true
- `(1, 2) != (2, 3)` is true

Note: = and != operators are not inverses of each other.

- `$a, $b`, and `$c` are bound to element nodes of type annotation xs:untypedAtomic, with string values "1", "2", and "2.0" respectively. Then `($a, $b) = ($c, 3.0)` returns false because $b and $c are compared as strings, but, `($a, $b) = ($c, 2.0)` returns true, because $b and 2.0 are compared as numbers.

Node Comparisons §3.5.3

`is <<(preceeds) >>(follows)`

Node comparisons are used to compare two nodes, by their identity or by their document order.

- The operands of a node comparison are evaluated in implementation-dependent order.
- If either operand is an empty sequence, the result of the comparison is an empty sequence.
- Each operand must be either a single node or an empty sequence; otherwise a type error is raised.
- A comparison with the `is` operator is true if the two operand nodes have the same identity, and are thus the same node; otherwise it is false. See [XQuery/XPath Data Model (XDM)] for a definition of node identity.
- A comparison with the << operator returns true if the left operand node precedes the right operand node in document order; otherwise it returns false.
- A comparison with the >> operator returns true if the left operand node follows the right operand node in document order; otherwise it returns false.
- `/books/book[isbn="1558604820"] is /books/book[call="QA76.9 C3845"]` true only if the left and right sides each evaluate to exactly the same single node
- `/transactions/purchase[parcel="28-451"] << /transactions/sale[parcel="33-870"]` true only if the node identified by the left side occurs before the node identified by the right side in document order.

12 Logical Expressions §3.6

`and or`

If a logical expression does not raise an error, its value is always one of the boolean values `true` or `false`.

- `1 eq 1 and 2 eq 2` is true
- `1 eq 1 or 2 eq 3` is true
- `1 eq 2 and 3 idiv 0 = 1` returns false or error in XPath 1.0 compatibility mode result is false
- `1 eq 1 or 3 idiv 0 = 1` returns true or error, in XPath 1.0 compatibility mode result is true
- `1 eq 1 and 3 idiv 0 = 1` returns an error

13 For Expressions §3.7

```
for $i in (10, 20),
    $j in (1, 2)
return ($i + $j)     result is a sequence of numbers: 11, 12, 21, 22
```

A variable bound in a `for` expression comprises all subexpressions of the `for` expression that appear after the variable binding. The scope does not include the expression to which the variable is bound.

The following example illustrates how a variable binding may reference another variable bound earlier in the same `for` expression:

```
     for $x in $z, $y in f($x)
         return g($x, $y)
```

The focus for evaluation of the `return` clause of a `for` expression is the same as the focus for evaluation of the `for` expression itself. Example:

- `fn:sum(for $i in order-item return @price * @qty)` find the total value of a set of order-items (incorrect)
- `fn:sum(for $i in order-item return $i/@price * $i/@qty)` find the total value of a set of order-items (correct)

14 Conditional Expressions §3.8

```
     if ($widget1/unit-cost < $widget2/unit-cost)
       then $widget1
       else $widget2
     if ($part/@discounted)
       then $part/wholesale
       else $part/retail
```

15 Quantified Expressions §3.9

`some every`

- `some`, the expression is true if at least one evaluation of the test expression has the effective boolean value true; otherwise the quantified expression is false.
- `every`, the expression is true if every evaluation of the test expression has the effective boolean value true; otherwise the quantified expression is false.
- `every $part in /parts/part satisfies $part/@discounted` `true` if every part element has a discounted attribute (regardless of the values of these attributes)
- `some $emp in /emps/employee satisfies ($emp/bonus > 0.25 * $emp/salary)` `true` if at least one employee element satisfies the given comparison expression
- `some $x in (1, 2, 3), $y in (2, 3, 4) satisfies $x + $y = 4` evaluates to true
- `every $x in (1, 2, 3), $y in (2, 3, 4) satisfies $x + $y = 4` evaluates to false
- `some $x in (1, 2, "cat") satisfies $x * 2 = 4` may either return true or raise a type error, since its test expression returns true for one variable binding and raises a type error for another
- `every $x in (1, 2, "cat") satisfies $x * 2 = 4` may either return false or raise a type error, since its test expression returns false for one variable binding and raises a type error for another

16 Expressions on SequenceTypes §3.10

Instance Of §3.10.1

The boolean operator `instance of` returns `true` if the value of its first operand matches the SequenceType in its second operand.

- `5 instance of xs:integer` returns `true`
- `5 instance of xs:decimal` returns `true` because `xs:integer` is derived by restriction from `xs:decimal`.
- `(5, 6) instance of xs:integer+` returns `true` because the given sequence contains two integers
- `. instance of element()` returns `true` if the context item is an element node or false if the context item is defined but is not an element node

Cast and Castable §3.10.2 and §3.10.3

The expression `V castable as T` returns `true` if the value `V` can be successfully cast into the target type `T` by using a `cast` expression; otherwise it returns `false`. The `castable` expression can be used as a predicate to avoid errors at evaluation time. It can also be used to select an appropriate type for processing of a given value, as illustrated in the following example:

```
if ($x castable as hatsize)
   then $x cast as hatsize
   else if ($x castable as IQ)
   then $x cast as IQ
   else $x cast as xs:string
```

Note: If the target type of a `castable` expression is `xs:QName`, or is a type that is derived from `xs:QName` or `xs:NOTATION`, and the input argument of the expression is of type `xs:string` but it is not a literal string, the result of the `castable` expression is `false`.

Constructor Functions §3.10.4

The name of the constructor function is the same as the name of its target type (except `xs:NOTATION` and `xs:anyAtomicType)` including namespace. The constructor function call `T($arg)` is defined to be equivalent to the expression `(($arg) cast as T?)`.

The constructor functions for `xs:QName` and for types derived from `xs:QName` and `xs:NOTATION` require their arguments to be string literals or to have a base type that is the same as the base type of the target type; otherwise a type error is raised.

- `xs:date("2000-01-01")` is equivalent to `("2000-01-01" cast as xs:date?)`
- `xs:decimal($floatvalue * 0.2E-5)` is equivalent to `(($floatvalue * 0.2E-5) cast as xs:decimal?)`
- `xs:dayTimeDuration("P21D")` returns a xs:dayTimeDuration value equal to 21 days. It is equivalent to `("P21D" cast as xs:dayTimeDuration?)`
- `usa:zipcode("12345")` is equivalent to the expression `("12345" cast as usa:zipcode?)`

An instance of an atomic type that is not in a namespace can be constructed in either of the following ways:

- `17 cast as apple`
- `apple(17)`

Treat §3.10.5

`treat` can be used to modify the static type of its operand.

Like `cast`, the `treat` expression takes two operands: an expression and a SequenceType. Unlike `cast`, however, `treat` does not change the dynamic type or value of its operand. Instead, the purpose of `treat` is to ensure that an expression has an expected dynamic type at evaluation time.

- `$myaddress treat as element(*, USAddress)` at run-time, the value of `$myaddress` must match the type `element(*, USAddress)`

HORIZONTAL AXIS

attribute

The attribute axis contains the attributes of the context node; the axis will be empty unless the context node is an element.
Examples:

- **attribute::name** selects the name attribute of the context node [@name]
- **attribute::*** selects all the attributes of the context node [@*]

following

The following axis contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants and excluding attribute nodes and namespace nodes.

following-sibling

The following-sibling axis contains all the following siblings of the context node; if the context node is an attribute node or namespace node, the following-sibling axis is empty.
Example:

- **following-sibling::chapter[position()=1]** selects the next chapter sibling of the context node

namespace

The namespace axis contains the namespace nodes of the context node; the axis will be empty unless the context node is an element.

preceding

The preceding axis contains all nodes in the same document as the context node that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes.

preceding-sibling

The preceding-sibling axis contains all the preceding siblings of the context node; if the context node is an attribute node or namespace node, the preceding-sibling axis is empty.
Example:

- **preceding-sibling::chapter[position()=1]** selects the previous chapter sibling of the context node

VERTICAL AXIS

ancestor

The ancestor axis contains the ancestors of the context node; the ancestors of the context node consist of the parent of context node and the parent's parent and so on; thus, the ancestor axis will always include the root node, unless the context node is the root node.
Example:

- **ancestor::div** selects all div ancestors of the context node

ancestor-or-self

The ancestor-or-self axis contains the context node and the ancestors of the context node; thus, the ancestor axis will always include the root node.
Example:

- **ancestor-or-self::div** selects the div ancestors of the context node and, if the context node is a div element, the context node as well

child

The child axis contains the children of the context node.
Examples:

- **child::para** selects the para element children of the context node [para]
- **child::*** selects all element children of the context node [*]
- **child::text()** selects all text node children of the context node [text()]
- **child::node()** selects all the children of the context node, whatever their node type
- **child::chapter/descendant::para** selects the para element descendants of the chapter element children of the context node [chapter//para]
- **child::*/child::para** selects all para grandchildren of the context node [*/para]
- **child::para[position()=1]** selects the first para child of the context node [para[position()=1]]
- **child::para[position()=last()]** selects the last para child of the context node [para[position()=last()]]
- **child::para[position()=last()-1]** selects the last but one para child of the context node [para[position()=last()-1]]
- **child::para[position()>1]** selects all the para children of the context node other than the first para child of the context node [para[position()>1]]
- **/child::doc/child::chapter[position()=5]/child::section[position()=2]** selects the second section of the fifth chapter of the doc document element [/doc/chapter[5]/section[2]]
- **child::para[attribute::type="warning"]** selects all para children of the context node that have a type attribute with value warning [para[@type="warning"]]
- **child::para[attribute::type='warning'][position()=5]** selects the fifth para child of the context node that has a type attribute with value warning [para[@type="warning"][5]]
- **child::para[position()=5][attribute::type="warning"]** selects the fifth para child of the context node if that child has a type attribute with value warning [para[5] [@type="warning"]]
- **child::chapter[child::title='Introduction']** selects the chapter children of the context node that have one or more title children with string-value equal to Introduction [chapter[title="Introduction"]]
- **child::chapter[child::title]** selects the chapter children of the context node that have one or more title children [chapter[title]]
- **child::*[self::chapter or self::appendix]** selects the chapter and appendix children of the context node [chapter|children]
- **child::*[self::chapter or self::appendix][position()=last()]** selects the last chapter or appendix child of the context node [[chapter|appendix][position()=last()]]

descendant

The descendant axis contains the descendants of the context node; a descendant is a child or a child of a child and so on; thus the descendant axis never contains attribute or namespace nodes.
Example:

- **descendant::para** selects the para element descendants of the context node [.//para]
- **/descendant::para** selects all the para elements in the same document as the context node [//para]
- **/descendant::olist/child::item** selects all the item elements that have an olist parent and that are in the same document as the context node [//olist/item]
- **/descendant::figure[position()=42]** selects the forty-second figure element in the document [//figure[position()=42]]

descendant-or-self

The descendant-or-self axis contains the context node and the descendants of the context node.
Example:

- **descendant-or-self::para** selects the para element descendants of the context node and, if the context node is a para element, the context node as well

parent

The parent axis contains the parent of the context node, if there is one.
Example:

- **parent::** select the parent element [..]
- **/** selects the document root (which is always the parent of the document element) [//]

self

The self axis contains just the context node itself.
Example:

- **self::para** selects the context node if it is a para element, and otherwise selects nothing [.]

NODE SET FUNCTIONS

number count**(node-set?)** § 4.1

The count function returns the number of nodes in the argument node-set.

node-set id**(object)** § 4.1

The id function selects elements by their unique ID.

number last**()** § 4.1

The last function returns a number equal to the context size from the expression evaluation context.

string local-name**(node-set?)** § 4.1

The local-name function returns the local part of the expanded-name of the node in the argument node-set that is first in document order.

string name**(node-set?)** § 4.1

The name function returns a string containing a QName representing the expanded-name of the node in the argument node-set that is first in document order.

string namespace-uri**(node-set?)** § 4.1

The namespace-uri function returns the namespace URI of the expanded-name of the node in the argument node-set that is first in document order.

number position**()** § 4.1

The position function returns a number equal to the context position from the expression evaluation context.

STRING FUNCTIONS

string concat**(string, string, string*)** § 4.2

The concat function returns the concatenation of its arguments.

boolean contains**(string, string)** **§ 4.2**

The contains function returns true if the first argument string contains the second argument string, and otherwise returns false.

string normalize-space**(string?)** **§ 4.2**

The normalize-space function returns the argument string with whitespace normalized by stripping leading and trailing whitespace and replacing sequences of whitespace characters by a single space.

boolean starts-with**(string, string)** **§ 4.2**

The starts-with function returns true if the first argument string starts with the second argument string, and otherwise returns false.

string string**(object?)** **§ 4.2**

The string function converts an object to a string:

- NaN is converted to the string NaN,
- positive zero is converted to the string 0,
- negative zero is converted to the string 0,
- positive infinity is converted to the string Infinity,
- negative infinity is converted to the string –Infinity,
- if the number is an integer, the number is represented in decimal form as a Number with no decimal point and no leading zeros, preceded by a minus sign (-) if the number is negative,
- otherwise, the number is represented in decimal form as a Number including a decimal point with at least one digit before the decimal point and at least one digit after the decimal point, preceded by a minus sign (-) if the number is negative.

number string-length**(string?)** **§ 4.2**

The string-length returns the number of characters in the string

string substring**(string, number, number?)** **§ 4.2**

The substring function returns the substring of the first argument starting at the position specified in the second argument with length specified in the third argument.

string substring-after**(string, string)** **§ 4.2**

The substring-after function returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string.

string substring-before**(string, string)** **§ 4.2**

The substring-before function returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string.

string translate**(string, string, string)** **§ 4.2**

The translate function returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.

BOOLEAN FUNCTIONS

boolean boolean**(object)** **§ 4.3**

The boolean function converts its argument to a Boolean:

- a number is true if and only if it is neither positive or negative zero nor NaN,
- a node-set is true if and only if it is non-empty,
- a string is true if and only if its length is non-zero,
- an object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type.

boolean false**()** **§ 4.3**

The false function returns false.

boolean lang**(string)** **§ 4.3**

The lang function returns true or false depending on whether the language of the context node as specified by *xml:lang* attributes is the same as or is a sublanguage of the language specified by the argument string.

boolean not**(boolean)** **§ 4.3**

The not function returns true if its argument is false, and false otherwise.

boolean true**()** **§ 4.3**

The true function returns true.

NUMBER FUNCTIONS

number ceiling**(number)** **§ 4.4**

The ceiling function returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer.

number floor**(number)** **§ 4.4**

The floor function returns the largest (closest to positive infinity) number that is not greater than the argument and that is an integer.

number number**(object?)** **§ 4.4**

The number function converts its argument to a number:
a string that consists of optional whitespace followed by an optional minus sign followed by a Number followed by whitespace is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string; any other string is converted to NaN,

- boolean true is converted to 1; boolean false is converted to 0,
- a node-set is first converted to a string as if by a call to the string function and then converted in the same way as a string argument,
- an object of a type other than the four basic types is converted to a number in a way that is dependent on that type.

number round**(number)** **§ 4.4**

The round function returns the number that is closest to the argument and that is an integer.

number sum**(node-set)** **§ 4.4**

The sum function returns the sum, for each node in the argument node-set, of the result of converting the string-values of the node to a number.

Quick Reference

XML Path Language (XPath)

Version 1.0

W3C Recommendation
16 November 1999

`http://www.w3.org/TR/xpath/`

Table of Contents:

deepX Ltd.

Dublin, Ireland

info@deepX.com
http://www.deepX.com/

XML Schema - Data Types Quick Reference

ver 1/03

©2002, 2003 D Vint Productions
xmlhelp@dvint.com
http://www.xml.dvint.com

1 Namespaces §3.1 pt2

- http://www.w3.org/2001/XMLSchema
- http://www.w3.org/2001/XMLSchema-datatypes

2 Logic Types

boolean	atomic	binary-valued logic legal literals {true, false, 1, 0}	§3.2.1.2 pt2

3 Binary Data Types

base64Binary	atomic	Base64-encoded arbitrary binary data.	§3.2.16 pt2
hexBinary	atomic	Arbitrary hex-encoded binary data. Example, “0FB7” is a hex encoding for 16-bit int 4023 (binary 111110110111).	§3.2.15 pt2

4 Text types

anyURI	atomic	A Uniform Resource Identifier Reference (URI). Can be absolute or relative, and may have an optional fragment identifier.	§3.2.17 pt2
language	derived token	natural language identifiers [RFC 1766] Example: en, fr.	§3.3.3 pt2
normalizedString	derived string	White space normalized strings	§3.3.1 pt2
string	atomic	Character strings in XML	§3.2.1 pt2
token	derived normalized-String	Tokenized strings.	§3.3.2 pt2

5 Number Types

byte	derived short	127 to-128. Sign is omitted, “+” assumed. Example: -1, 0, 126, +100.	§3.3.19 pt2
decimal	atomic	Arbitrary precision decimal numbers. Sign omitted, “+” is assumed. Leading and trailing zeroes are optional. If the fractional part is zero, the period and following zero(es) can be omitted.	§3.2.3 pt2
double	atomic	Double-precision 64-bit floating point type - legal literals {0, -0, INF, -INF and NaN} Example, -1E4, 12.78e-2, 12 and INF	§3.2.5 pt2
float	atomic	32-bit floating point type - legal literals {0, -0, INF, -INF and NaN} Example, -1E4, 1267.43233E12, 12.78e-2, 12 and INF	§3.2.4 pt2
int	derived long	2147483647 to -2147483648. Sign omitted, “+” is assumed. Example: -1, 0, 126789675, +100000.	§3.3.17 pt2
integer	derived decimal	Integer or whole numbers - Sign omitted, “+” is assumed. Example: -1, 0, 12678967543233, +100000.	§3.3.13 pt2
long	derived integer	9223372036854775807 to -9223372036854775808. Sign omitted, “+” assumed. Example: -1, 0, 12678967543233, +100000.	§3.3.16 pt2
negativeInteger	derived nonPositive	Infinite set {...,-2,-1}. Example: -1, -12678967543233, -100000.	§3.3.15 pt2
nonNegativeInteger	derived integer	Infinite set {0, 1, 2,...}. Sign omitted, “+” assumed. Example: 1, 0, 12678967543233, +100000.	§3.3.20 pt2
nonPositiveInteger	derived integer	Infinite set {...,-2,-1,0}. Example: -1, 0, -126733, -100000.	§3.3.14 pt2
positiveInteger	derived nonNegativeInteger	Infinite set {1, 2,...}. Optional “+” sign,. Example: 1, 12678967543233, +100000.	§3.3.25 pt2
short	derived int	32767 to -32768. Sign omitted, “+” assumed. Example: -1, 0, 12678, +10000.	§3.3.18 pt2
unsignedByte	derived unsignedShort	0 to 255. a finite-length Example: 0, 126, 100.	§3.3.24 pt2
unsignedInt	derived unsignedLong	0 to 4294967295 Example: 0, 1267896754, 100000.	§3.3.22 pt2
unsignedLong	derived nonNegative	0 to 18446744073709551615. Example: 0, 12678967543233, 100000.	§3.3.21 pt2
unsignedShort	derived unsignedInt	0 to 65535. Example: 0, 12678, 10000.	§3.3.23 pt2

6 Date Time Types

date	atomic	Calendar date.Format CCYY-MM-DD. Example, May the 31st, 1999 is: 1999-05-31.	§3.2.9 pt2
dateTime	atomic	Specific instant of time. ISO 8601 extended format CCYY-MM-DDThh:mm:ss. Example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC): 1999-05-31T13:20:00-05:00.	§3.2.7 pt2
duration	atomic	A duration of time. ISO 8601 extended format PnYn MnDTnH nMn S. Example, to indicate duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes: P1Y2M3DT10H30M. One could also indicate a duration of minus 120 days as: -P120D.	§3.2.6 pt2
gDay	atomic	Gregorian day. Example a day such as the 5th of the month is --05.	§3.2.13 pt2
gMonth	atomic	Gregorian month. Example: May is --05--	§3.2.14 pt2
gMonthDay	atomic	Gregorian specific day in a month. Example: Feb 5 is --02-05.	§3.2.12 pt2
gYear	atomic	Gregorian calendar year. Example, year 1999, write: 1999.	§3.2.11 pt2
gYearMonth	atomic	Specific gregorian month and year. Example, May 1999, write: 1999-05.	§3.2.10 pt2
time	atomic	An instant of time that recurs every day. Example, 1:20 pm for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), write: 13:20:00-05:00.	§3.2.8 pt2

7 XML Types

Name	derived token	XML Names	§3.3.6 pt2
NCName	derived Name	XML “non-colonized” Names.	§3.3.7 pt2
NOTATION	atomic	NOTATION type	§3.2.19 pt2
QName	atomic	XML qualified names	§3.2.18 pt2

Following types should only be used in attribute declaration for XML compatibility:

ENTITY	derived NCName	ENTITY attribute type	§3.3.11 pt2
ENTITIES	derived ENTITY	ENTITIES attribute type	§3.3.12 pt2
ID	derived NCNAME	ID attribute type	§3.3.8 pt2
IDREF	derived NCName	IDREF attribute type	§3.3.9 pt2
IDREFS	derived IDREF	IDREFS attribute type	§3.3.10 pt2
NMTOKEN	derived token	NMTOKEN attribute type	§3.3.4 pt2
NMTOKENS	derived NMTOKENS	NMTOKENS attribute type	§3.3.5 pt2

8 Built-in Types

anyType	ur-type definition	Built-in Complex type definition of Ur-Type.	§3.4.7 pt1
anySimpleType	ur-type definition	Built-in Simple type definition of Ur-Type.	§4.1.6 pt2

9 Simple Data Type Declaration §4.1.2 pt2

Note: All schema components allow attributes from non-schema namespaces.

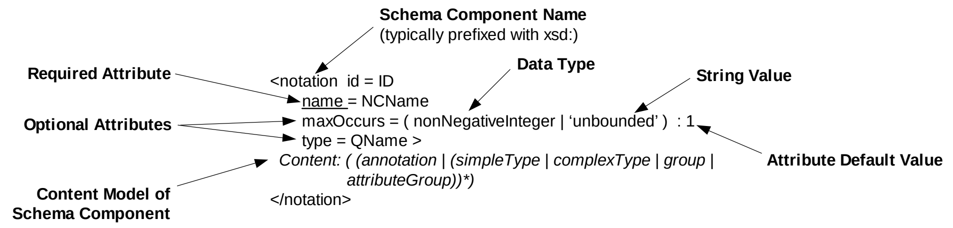
<simpleType id = ID
final = ('#all' | ('list' | 'union' | 'restriction'))
<u>name</u> = NCName>
Content: (annotation ?, (restriction | list | union)) </simpleType>

<list id = ID
itemType = QName>
Content: (annotation ?, (simpleType ?)) </list>

<union id = ID
memberTypes = List of QName>
*Content: (annotation ?, (simpleType *))* </union>

<restriction id = ID
<u>base</u> = QName>
Content: (annotation ?, (simpleType ?, (minExclusive | minInclusive | maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength | enumeration | whiteSpace | pattern)))* </restriction>

.



```
<notation  id = ID
  name = NCName
  maxOccurs = ( nonNegativeInteger | ‘unbounded’ )  : 1
  type = QName >
 Content: ( (annotation | (simpleType | complexType | group |
            attributeGroup))*)
</notation>
```

10 Constraining Facets §4.3 pt2

<length id = ID
fixed = boolean : false
value = nonNegativeInteger >
Content: (annotation?) </length>

<minLength id = ID
fixed = boolean : false
value = nonNegativeInteger >
Content: (annotation?) </minLength>

<maxLength id = ID
fixed = boolean : false
value = nonNegativeInteger >
Content: (annotation?) </maxLength>

<pattern id = ID
value = anySimpleType >
Content: (annotation?) </pattern>

<enumeration id = ID
value = anySimpleType >
Content: (annotation?) </enumeration>

<whiteSpace id = ID
fixed = boolean : false
value = ('collapse '| 'preserve '|
'replace ') >
Content: (annotation?) </whitespace>

<maxInclusive id = ID
fixed = boolean : false
value = anySimpleType >
Content: (annotation?) </maxInclusive>

<maxExclusive id = ID
fixed = boolean : false
value = anySimpleType >
Content: (annotation?) </maxExclusive>

<minInclusive id = ID
fixed = boolean : false
value = anySimpleType />
Content: (annotation?) </minInclusive>

<minExclusive id = ID
fixed = boolean : false
value = anySimpleType >
Content: (annotation?) </minExclusive>

<totalDigits id = ID
fixed = boolean : false
value = positiveInteger >
Content: (annotation?) </totalDigits>

<fractionDigits id = ID
fixed = boolean : false
value = nonNegativeInteger >
Content: (annotation?) </fractionDigits>

11 Simple Data Types and Constraining Facets §4.1.5 pt2, Appendix B pt0

Simple Data Type	length	minLength	maxLength	pattern	enumeration	whiteSpace	maxInclusive	maxExclusive	minExclusive	minInclusive	totalDigits	fractionDigits
anyURI	u	u	u	u	u	u						
base64Binary	u	u	u	u	u	u						
boolean				u		u						
byte - 127 to-128				u	u	u	u	u	u	u	u	u
date - CCYY-MM-DD				u	u	u	u	u	u	u		
dateTime - CCYY-MM-DDThh:mm:ss				u	u	u	u	u	u	u		
decimal - Arbitrary precision decimal numbers				u	u	u	u	u	u	u	u	u
double - Double-precision 64-bit floating point				u	u	u	u	u	u	u		
duration - PnYn MnDTnH nMn S				u	u	u	u	u	u	u		
ENTITIES	u	u	u		u	u						
ENTITY	u	u	u	u	u	u						
float - 32-bit floating point type				u	u	u	u	u	u	u		
gDay				u	u	u	u	u	u	u		
gMonth				u	u	u	u	u	u	u		
gMonthDay				u	u	u	u	u	u	u		
gYear				u	u	u	u	u	u	u		
gYearMonth				u	u	u	u	u	u	u		
hexBinary	u	u	u	u	u	u						
ID	u	u	u	u	u	u						
IDREF	u	u	u	u	u	u						
IDREFS	u	u	u		u	u						
int - 2147483647 to -2147483648.				u	u	u	u	u	u	u	u	u
integer				u	u	u	u	u	u	u	u	u
language - RFC 1766] Example: en, fr	u	u	u	u	u	u						
list	u	u	u	u	u	u						
long - 9223372036854775807 to - 9223372036854775808				u	u	u	u	u	u	u	u	u
Name	u	u	u	u	u	u						
NCName	u	u	u	u	u	u						
negativeInteger				u	u	u	u	u	u	u	u	u
NMTOKEN	u	u	u	u	u	u						
NMTOKENS	u	u	u		u	u						
nonNegativeInteger				u	u	u	u	u	u	u	u	u
nonPositiveInteger				u	u	u	u	u	u	u	u	u
normalizedString	u	u	u	u	u	u						
NOTATION	u	u	u	u	u	u						
positiveInteger				u	u	u	u	u	u	u	u	u
QName	u	u	u	u	u	u						
short - 32767 to -32768				u	u	u	u	u	u	u	u	u
string	u	u	u	u	u	u						
time - hh:mm:ss				u	u	u	u	u	u	u		
token	u	u	u	u	u	u						
union				u	u							
unsignedByte - 0 to 255				u	u	u	u	u	u	u	u	u
unsignedInt - 0 to 4294967295				u	u	u	u	u	u	u	u	u
unsignedLong - 0 to 18446744073709551615				u	u	u	u	u	u	u	u	u
unsignedShort - 0 to 65535				u	u	u	u	u	u	u	u	u

12 Regular Expressions for Pattern Facet §4.3.4 pt2 §Appendix D pt0, §Appendix F pt2

Special Characters needing to be escaped with a '\'

\ | . - ^ ? * + { } ( ) [ ]

Character References

&#x4E; or &#99; for hex or decimal XML character references

Repetition Operators

*	0 or more,
?	0 or 1,
+	1 or more

Interval Operators

{x,y} range x to y, {x,} at least x, {x} exactly x, i.e. {4,8} 4 to 8

Character Range Expressions

[a-zA-Z] = character a to z upper and lower case
[0-9] = digits 0 to 9

Special Character Sequences

\n	newline
\r	return
\t	tab
. (dot)	all characters except newline and return
\s	space characters (space, tab, newline, return)
\S	non-Space characters
\i	initial XML name characters (letter _ ;)
\I	not initial XML name characters
\c	XML NameChar characters
\C	not XML NameChar characters
\d	decimal digits
\D	not decimal digits
\w	XML Letter or Digit characters
\W	not XML Letter or Digit characters
\p{IsBasicLatin}	block escape identifying ASCII characters, similar IsGreek, IsHebrew, IsThai for these ranges of Unicode blocks
\p{L}	all Letters
\p{M}	all Marks
\p{N}	all Numbers
\p{P}	all Punctuation
\p{Z}	all Separators
\p{S}	all Symbols

\p{C}	all Others. Additional modifying values like Lu = uppercase,
	Ll = lowercase, Nd = decimal digit, Sm = math symbols,
	Sc = currency

\P{}	not the block or category, \P{IsGreek} = not Greek block

Pattern Examples

Expression	Match(es)
Chapter \d	Chapter 0, Chapter 1, Chapter 2....
Chapter\s\w	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a word character (XML 1.0 Letter or Digit)
Espan&#xF1;ola	Española
\p{Lu}	any uppercase character, the value of \p{} (e.g. “Lu”) is defined by Unicode
a*x	x, ax, aax, aaax....
a?x	ax, x
a+x	ax, aax, aaax....
(a|b)+x	ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax....
[^0-9]x	any non-digit character followed by the character x
\Dx	any non-digit character followed by the character x
.x	any character followed by the character x
.*abc.*	1x2abc, abc1x2, z3456abchooray....
ab{2}x	abbx
ab{2,4}x	abbx, abbbx, abbbbx
ab{2,}x	abbx, abbbx, abbbbx....
(ab){2}x	ababx

ver 1/03

©2002, 2003 D Vint Productions
xmlhelp@dvint.com
http://www.xml.dvint.com

[redacted]

XML Schema - Structures Quick Reference

ver 1/03

©2002, 2003 D Vint Productions
xmlhelp@dvint.com
http://www.xml.dvint.com

Note: All schema components allow attributes from non-schema namespaces.

1 Namespaces §2.6 pt1

- http://www.w3.org/2001/XMLSchema
- http://www.w3.org/2001/XMLSchema-instance

2 Schema Declaration §3.15.2 pt1

```
<schema  id = ID
  attributeFormDefault = ( ‘qualified’| ‘unqualified’) : ‘unqualified”
  blockDefault = ( ‘#al l | List of ( ‘extension’| ‘restriction’| ‘substitution’))  : ''
  elementFormDefault = ( ‘qualified’ |  ‘unqualified’) : ‘unqualified’
  finalDefault = ( ‘#all’| List of ( ‘extension’| ‘restriction’))  : ''
  targetNamespace = anyURI
  version = token
  xml:lang = language >
  Content: ((include | import | redefine | annotation)*, (((simpleType | complexType |
            group | attributeGroup) | element | attribute | notation), annotation*)*)
</schema>
```

3 Schema Management §4.2.1, 4.2.2, 4.2.3 pt1

```
<include  id = ID
  schemaLocation = anyURI >
  Content: (annotation?) </include>

<redefine  id = ID
  schemaLocation = anyURI>
  Content: (annotation | (simpleType | complexType | group | attributeGroup))* </redefine>

<import  id = ID
  namespace = anyURI
  schemaLocation = anyURI>
  Content: (annotation?) </import>
```

4 Simple Data Type Declaration §3.14.2 pt1 and §4.1.2 pt2

```
<simpleType   id = ID
  final = ( ‘#all’| ( ‘list’| ‘union’| ‘restriction’))
  name = NCName>
  Content: ( annotation ?, ( restriction | list | union )) </simpleType>

<list  id = ID
  itemType = QName>
   Content: ( annotation ?, ( simpleType ?)) </list>

<union   id = ID
  memberTypes = List of QName>
   Content: ( annotation ?, ( simpleType *)) </union>

<restriction   id = ID
  base = QName>
  Content: ( annotation ?, ( simpleType ?, ( minExclusive | minInclusive |
     maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength |
    maxLength | enumeration | whiteSpace | pattern )*)) </restriction>
```

Constraining Facets §4.3 pt2

```
<length   id = ID
  fixed = boolean : false
  value = nonNegativeInteger >
Content: (annotation?) </length>

<minLength  id = ID
  fixed = boolean : false
  value = nonNegativeInteger >
Content: (annotation?) </minLength>

<maxLength  id = ID
  fixed = boolean : false
  value = nonNegativeInteger >
Content: (annotation?) </maxLength>

<pattern  id = ID
  value = anySimpleType >
Content: (annotation?) </pattern>

<enumeration  id = ID
  value = anySimpleType >
Content: (annotation?) </enumeration>

<whiteSpace  id = ID
  fixed = boolean : false
  value = ( ‘collapse’| ‘preserve’|
‘replace’) >
Content: (annotation?) </whitespace>

<maxInclusive  id = ID
  fixed = boolean : false
  value = anySimpleType >
Content: (annotation?) </maxInclusive>

<maxExclusive  id = ID
  fixed = boolean : false
  value = anySimpleType >
Content: (annotation?) </maxExclusive>

<minInclusive  id = ID
  fixed = boolean : false
  value = anySimpleType />
Content: (annotation?) </minInclusive>

<minExclusive  id = ID
  fixed = boolean : false
  value = anySimpleType >
Content: (annotation?) </minExclusive>

<totalDigits  id = ID
  fixed = boolean : false
  value = positiveInteger >
Content: (annotation?) </totalDigits>

<fractionDigits  id = ID
  fixed = boolean : false
  value = nonNegativeInteger >
Content: (annotation?) </fractionDigits>
```

5 Complex Data Type Declaration §3.4.2 pt1

```
<complexType  id = ID
  abstract = boolean :  ‘false’
  block = ( ‘#all’| List of ( ‘extension’|  ‘restriction’))
  final = ( ‘#all’| List of ( ‘extension’| ‘restriction’))
  mixed = boolean :  ‘false’
  name = NCName >
  Content: (annotation?, (simpleContent | complexContent | ((group | all | choice |
       sequence)?, ((attribute | attributeGroup)*, anyAttribute?)))) </complexType>
```

Simple Content §3.4.2 pt1

```
<simpleContent  id = ID>
  Content: (annotation?, (restriction | extension)) </simpleContent>

<restriction   id = ID
  base = QName>
  Content: (annotation?, (simpleType?, (minExclusive | minInclusive | maxExclusive
       | maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength |
        enumeration | whiteSpace | pattern)*)?, ((attribute | attributeGroup)*,
        anyAttribute?)) </restriction>

<extension  id = ID
  base = QName>
  Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?)) </extension>
```

Complex Content §3.4.2 pt1

```
<complexContent  id = ID
  mixed = boolean>
  Content: (annotation?, (restriction | extension)) </complexContent>

<restriction  id = ID
  base = QName>
  Content: (annotation?, (group | all | choice | sequence)?,
       ((attribute | attributeGroup)*, anyAttribute?)) </restriction>

<extension  id = ID
  base = QName>
  Content: (annotation?, ((group | all | choice | sequence)?,
      ((attribute | attributeGroup)*, anyAttribute?))) </extension>
```

6 Element Declaration §3.3.2 pt1

```
<element  id = ID
  abstract = boolean :  ‘false’
  block = ( ‘#all’| List of ( ‘extension’| ‘restriction’| ‘substitution’))
  default = string
  final = ( ‘#all’| List of ( ‘extension’| ‘restriction’))
  fixed = string
  form = ( ‘qualified’| ‘unqualified’)
  maxOccurs = (nonNegativeInteger | ‘unbounded’)  : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = boolean :  ‘false’
  ref = QName
  substitutionGroup = QName
  type = QName >
  Content: (annotation?, ((simpleType | complexType)?,
        (unique | key | keyref)*)) </element>
```

7 Content Model §3.8.2 pt1

```
<choice  id = ID
  maxOccurs = (nonNegativeInteger | ‘unbounded’)  : 1
  minOccurs = nonNegativeInteger : 1}>
  Content: (annotation?, (element | group | choice | sequence | any)*) </choice>

<sequence  id = ID
  maxOccurs = (nonNegativeInteger | ‘unbounded’)  : 1
  minOccurs = nonNegativeInteger : 1}>
  Content: (annotation?, (element | group | choice | sequence | any)*) </sequence>

<all  id = ID
  maxOccurs = 1 : 1  minOccurs = (0 | 1) : 1>
  Content: (annotation?, element*) </all>
```

8 Wildcard Schema Component §3.10.2 pt1

```
<any  id = ID
  maxOccurs = ( nonNegativeInteger | ‘unbounded’)  : 1
  minOccurs = nonNegativeInteger : 1
  namespace = (( ‘##any’| ‘##other’) | List of (anyURI | ( ‘##targetNamespace’| ‘##local’)) )  : ‘##any’
  processContents = ( ‘lax’| ‘skip’| ‘strict’) : ‘strict’>
  Content: (annotation?) </any>
```

§3.4.2 pt1

```
<anyAttribute  id = ID
  namespace = (( ‘##any’| ‘##other’) | List of ( anyURI | ( ‘##targetNamespace’| ‘##local’)) )  : ‘##any’
  processContents = ( ‘lax’| ‘skip’| ‘strict’) : ‘stric t >
  Content: (annotation?) </anyAttribute>
```

9 Attribute Declaration §3.2.2 pt1

```
<attribute  id = ID
  default = string
  fixed = string
  form = ( ‘qualified’| ‘unqualified’)
  name = NCName
  ref = QName
  type = QName
  use = ( ‘optional’| ‘prohibited’| ’required’) : ‘optional’ >
  Content: (annotation?, (simpleType?)) </attribute>
```

10 Element Group Declaration (*parameter entity like*) §3.7.2 pt1

<group id = ID
maxOccurs = (nonNegativeInteger | ‘unbounded ’) : 1
minOccurs = nonNegativeInteger : 1
name = NCName
ref = QName >
Content: (annotation?, (all | choice | sequence)?) </group>

11 Attribute Group Declaration (*parameter entity like)* §3.6.2 pt1

<attributeGroup id = ID
name = NCName
ref = QName >
Content: (annotation?, ((attribute | attributeGroup), anyAttribute?))* </attributeGroup>

12 Identity-constraint Definitions §3.11.2 pt1

<unique id = ID
name = NCName >
Content: (annotation?, (selector, field+)) </unique>

<key id = ID
name = NCName >
Content: (annotation?, (selector, field+)) </key>

<keyref id = ID
name = NCName
refer = QName >
Content: (annotation?, (selector, field+)) </keyref>

<selector id = ID
xpath = a subset of XPath expression >
Content: (annotation?) </selector>

<field id = ID
xpath = a subset of XPath expression >
Content: (annotation?) </field>

13 Schema Documentation Components §3.13.2 pt1

<annotation id = ID>
*Content: (appinfo | documentation)** </annotation>

<appinfo
source = anyURI>
*Content: ({any})** </appinfo>

<documentation
source = anyURI
xml:lang = language>
*Content: ({any})** </documentation>

14 Notation Declaration §3.12.2 pt1

<notation id = ID
name = NCName
public = anyURI
system = anyURI >
Content: (annotation?) </notation>

15 Defined Attribute Values

{any} Any element not part of Schema namespace.

#all All of the values listed

[final attribute] *controls further derivation* §3.4.1 pt1

list A finite-length (possibly empty) sequence of values

union A combination of the of one or more other datatypes.

restriction Values for constraining facets are specified to a subset of those of its base type.

[namespace attribute] *controls use of namespaces* §3.4.2 pt1

##any Any namespace (default)

##other Any namespace other than target namespace

##targetNamespace Must belong to the target namespace of schema

##local Any unqualified XML from local namespace

[processContents attribute] *specify how contents should be processed for validation* §3.10.1 pt1

strict There must be a top-level declaration for the item available, or the item must have an xsi:type, and must be valid.

skip No constraints at all: the item must simply be well-formed.

lax Validate where you can, don't worry when you can't.

[form attribute] *controls namespace qualifying* §3.2.2 pt1

qualified Namespace qualified

unqualified No namespace qualification

[use attribute] *specifies the use of an attribute* §3.2.2 pt1

optional Attribute is optional

prohibited Attribute is prohibited

required Attribute is required to have a value

[whitespace attribute] *specifies whitespace handling* §3.1.4 pt 1, §4.3.6 pt 2

preserve The value is the normalized value

replace All occurrences of tab, line feed and carriage return are replaced with space.

collapse Contiguous sequences of spaces are collapsed to a single space, and initial and/or final spaces are deleted.

16 Built-in Types

anyType Built-in Complex type definition of Ur-Type. §3.4.7 pt1

anySimpleType Built-in Simple type definition of Ur-Type. §3.14.7 pt1

17 Schema Instance Related Markup §2.6 pt1 and §3.2.7 pt1

xsi:type An element in an instance may explicitly assert its type using the attribute xsi:type. The value is a QName associated with a type definition. §2.6.1 pt1

xsi:nil An element may be valid without content if it has the attribute xsi:nil with the value true. §2.6.2 pt1

xsi:noNamespaceSchemaLocation, xsi:schemaLocation Provide hints as to the physical location of schema documents §2.6.3 pt1

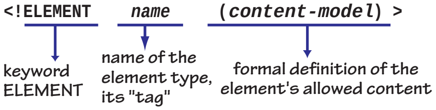
18 Simple Data Types and Constraining Facets

Simple Data Type	length	minLength	maxLength	pattern	enumeration	whiteSpace	maxInclusive	maxExclusive	minExclusive	minInclusive	totalDigits	fractionDigits
anyURI	u	u	u	u	u	u						
base64Binary	u	u	u	u	u	u						
boolean				u		u						
byte - 127 to-128				u	u	u	u	u	u	u	u	u
date - CCYY-MM-DD				u	u	u	u	u	u	u		
dateTime - CCYY-MM-DDThh:mm:ss				u	u	u	u	u	u	u		
decimal - Arbitrary precision decimal numbers				u	u	u	u	u	u	u	u	u
double - Double-precision 64-bit floating point				u	u	u	u	u	u	u		
duration - PnYn MnDTnH nMn S				u	u	u	u	u	u	u		
ENTITIES	u	u	u		u	u						
ENTITY	u	u	u	u	u	u						
float - 32-bit floating point type				u	u	u	u	u	u	u		
gDay				u	u	u	u	u	u	u		
gMonth				u	u	u	u	u	u	u		
gMonthDay				u	u	u	u	u	u	u		
gYear				u	u	u	u	u	u	u		
gYearMonth				u	u	u	u	u	u	u		
hexBinary	u	u	u	u	u	u						
ID	u	u	u	u	u	u						
IDREF	u	u	u	u	u	u						
IDREFS	u	u	u		u	u						
int - 2147483647 to -2147483648.				u	u	u	u	u	u	u	u	u
integer				u	u	u	u	u	u	u	u	u
language - RFC 1766] Example: en, fr	u	u	u	u	u	u						
list	u	u	u	u	u	u						
long - 9223372036854775807 to -9223372036854775808				u	u	u	u	u	u	u	u	u
Name	u	u	u	u	u	u						
NCName	u	u	u	u	u	u						
negativeInteger				u	u	u	u	u	u	u	u	u
NMTOKEN	u	u	u	u	u	u						
NMTOKENS	u	u	u		u	u						
nonNegativeInteger				u	u	u	u	u	u	u	u	u
nonPositiveInteger				u	u	u	u	u	u	u	u	u
normalizedString	u	u	u	u	u	u						
NOTATION	u	u	u	u	u	u						
positiveInteger				u	u	u	u	u	u	u	u	u
QName	u	u	u	u	u	u						
short - 32767 to -32768				u	u	u	u	u	u	u	u	u
string	u	u	u	u	u	u						
time - hh:mm:ss				u	u	u	u	u	u	u		
token	u	u	u	u	u	u						
union				u	u							
unsignedByte - 0 to 255				u	u	u	u	u	u	u	u	u
unsignedInt - 0 to 4294967295				u	u	u	u	u	u	u	u	u
unsignedLong - 0 to 18446744073709551615				u	u	u	u	u	u	u	u	u
unsignedShort - 0 to 65535				u	u	u	u	u	u	u	u	u

ver 1/03

xmlhelp@dvint.com
http://www.xml.dvint.com

Element Declaration


`<!ELEMENT name (content-model) >`

keyword ELEMENT

name of the element type, its "tag"

formal definition of the element's allowed content

Connectors

,	*"Then"*	Follow with (in sequence)
|	*"Or"*	Select (only) one from the group
Only one connector type per group — no mixing!		

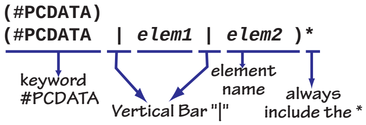
Occurrence Indicators

(no indicator)	*Required*	One and only one
?	*Optional*	None or one
*	*Optional, repeatable*	None, one, or more
+	*Required, repeatable*	One or more

Groupings

(	Start content model or group
)	End content model or group

#PCDATA in Models (first, OR bars, asterisk)


`(#PCDATA)`
`(#PCDATA | elem1 | elem2 )*`

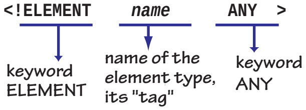
keyword #PCDATA

Vertical Bar "|"

element name

always include the *

ANY Element Keyword

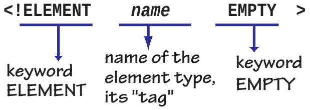

`<!ELEMENT name ANY >`

keyword ELEMENT

name of the element type, its "tag"

keyword ANY

EMPTY Element Keyword


`<!ELEMENT name EMPTY >`

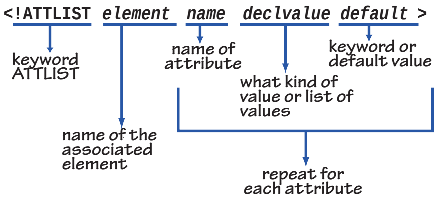
keyword ELEMENT

name of the element type, its "tag"

keyword EMPTY

Mulberry Technologies, Inc.

Attribute Declaration


`<!ATTLIST element name declvalue default >`

keyword ATTLIST

name of the associated element

name of attribute

what kind of value or list of values

keyword or default value

repeat for each attribute

Declared Value Keywords

CDATA	Data character string (default if well-formed)
NMTOKEN	Name token
NMTOKENS	One or more name tokens (spaces between)
ID	Unique identifier for element
IDREF	Reference to `ID` on another element
IDREFS	One or more `IDREF`s (spaces between)
ENTITY	Name of an entity (declared elsewhere)
ENTITIES	One or more names of entities

Enumerated Value Descriptions

(a|b|c)	List of attribute values (*Or* between)
NOTATION (x|y)	Names of notations (Requires a list of values as well as the keyword. Values declared elsewhere with NOTATION.)

Attribute Defaults

"*value*"	If attribute is omitted, assume this value.
#REQUIRED	Required. Document is *not valid* if no value is provided.
#IMPLIED	Optional. Not constrained; no default can be inferred; an application is free to handle as appropriate.
#FIXED "*value*"	Fixed value. (Requires a value as well as the keyword.) If the attribute appears with a different value, that's an error.

Reserved Attributes

xml:space	Preserve whitespace or use default
xml:lang	Indicate language of element and that element's attributes and children

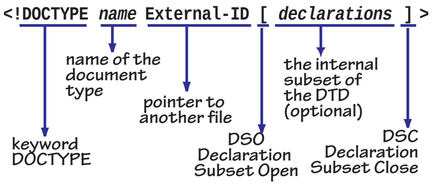
Mulberry Technologies, Inc.

XML Syntax Quick Reference

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.

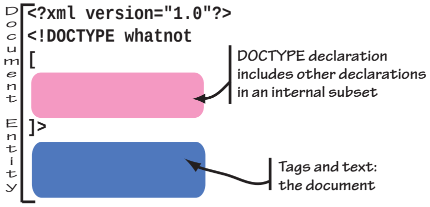
DOCTYPE Declaration



```
<!DOCTYPE name External-ID [ declarations ] >
```

name of the document type
pointer to another file
the internal subset of the DTD (optional)
keyword DOCTYPE
DSO Declaration Subset Open
DSC Declaration Subset Close

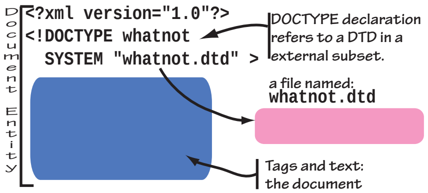
Internal Subset



```
<?xml version="1.0"?>
<!DOCTYPE whatnot
[
]>
```

Document Entity
DOCTYPE declaration includes other declarations in an internal subset
Tags and text: the document

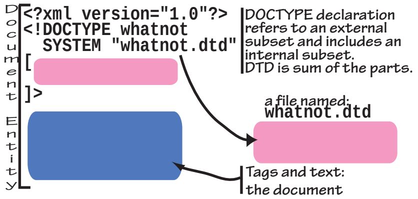
External Subset



```
<?xml version="1.0"?>
<!DOCTYPE whatnot
  SYSTEM "whatnot.dtd" >
```

Document Entity
DOCTYPE declaration refers to a DTD in a external subset.
a file named:
whatnot.dtd
Tags and text: the document

Internal and External Subsets



```
<?xml version="1.0"?>
<!DOCTYPE whatnot
  SYSTEM "whatnot.dtd"
[
]>
```

Document Entity
DOCTYPE declaration refers to an external subset and includes an internal subset.
DTD is sum of the parts.
a file named:
whatnot.dtd
Tags and text: the document

Conditional Section (DTD only)

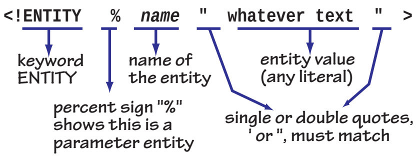
```
<![IGNORE[ declarations ]]>
<![INCLUDE[ declarations ]]>
```

External-ID

```
   SYSTEM "URI"
OR PUBLIC "Public ID" "URI"
```

Parameter Entity Declarations

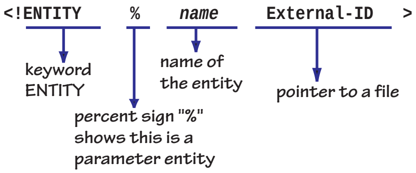
Internal Parameter Entity



```
<!ENTITY % name " whatever text " >
```

keyword ENTITY
percent sign "%" shows this is a parameter entity
name of the entity
entity value (any literal)
single or double quotes, ' or ", must match

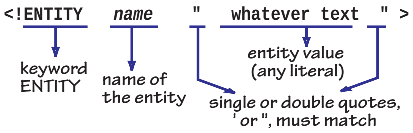
External Parameter Entity



```
<!ENTITY % name External-ID >
```

General Entity Declarations

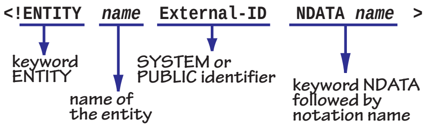
Internal Entity



```
<!ENTITY name " whatever text " >
```

keyword ENTITY
name of the entity
entity value (any literal)
single or double quotes, ' or ", must match

External Unparsed Entity



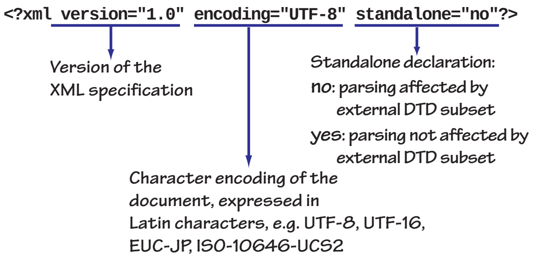
```
<!ENTITY name External-ID NDATA name >
```

keyword ENTITY
name of the entity
SYSTEM or PUBLIC identifier
keyword NDATA followed by notation name

Predefined General Entities

Entity	Displays As	Character Value
&amp;	&	&#38;#38;
&lt;	<	&#38;#60;
&gt;	>	&#62;
&apos;	'	&#39;
&quot;	"	&#34;

XML Declaration



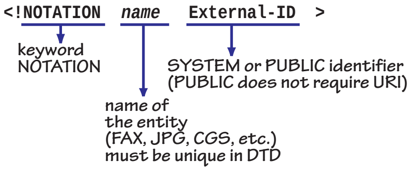
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Version of the XML specification
Standalone declaration:
no: parsing affected by external DTD subset
yes: parsing not affected by external DTD subset
Character encoding of the document, expressed in Latin characters, e.g. UTF-8, UTF-16, EUC-JP, ISO-10646-UCS2

Processing Instruction

```
<?target ***Some Stuff **** ?>
```

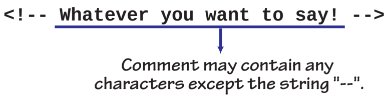
Notation Declaration



```
<!NOTATION name External-ID >
```

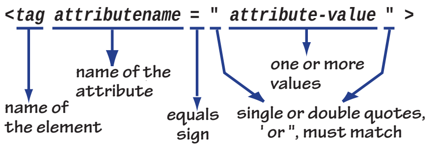
keyword NOTATION
name of the entity (FAX, JPG, CGS, etc.) must be unique in DTD
SYSTEM or PUBLIC identifier (PUBLIC does not require URI)

Comment



```
<!-- Whatever you want to say! -->
```

Start Tag with Attribute (in document)



```
<tag attributename = " attribute-value " >
```

EMPTY Element (in document)

```
<name/>
<name></name>
```

CDATA Section (in document)

```
<![CDATA[ *** Some Stuff *** ]]>
```

Mulberry Technologies, Inc.

Mulberry Technologies, Inc.

Mulberry Technologies, Inc.