XML Schema - Kurzreferenz

Schema

Root

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

Three Good Reasons for XML Schemas
- validate input file against Schema
- specification, documentation & validator
- OOP languages (Java/C#) directly allow to construct type hierarchies from XML Schemas

Assembling of schemas

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

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

<import
  id = ID
  namespace = anyURI
  schemaLocation = anyURI
  { foreign attributes }>
  Content: (annotation?)
</import>
http://www.w3.org/TR/xmlschema-1/#layer2
```

Elements

```
<element
  abstract = boolean : false
  block = (#all | List of (extension | restriction | substitution))
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  { foreign attributes }>
  Content: (annotation?, ((simpleType | complexType)?,
            (unique | key | keyref)*))
</element>
http://www.w3.org/TR/xmlschema-1/#cElement_Declarations
```

Element group

```
<group
  name = NCName>
  Content: (annotation?, (all | choice | sequence))
</group>
```

Attributes

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

http://www.w3.org/TR/xmlschema-1/#declare-attribute
```

Attribute group

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

http://www.w3.org/TR/xmlschema-1/#declare-attributeGroup
```

Types

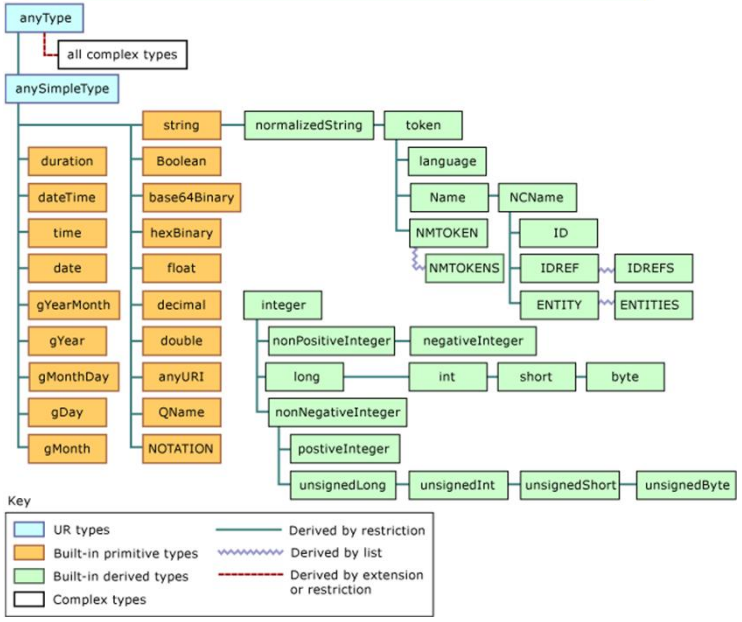
Global complex type

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

http://www.w3.org/TR/xmlschema-1/#declare-type
Derivation

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

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

Global simple type

```
<simpleType
  final = (#all | List of (list | union | restriction))
  id = ID
  name = NCName
  { foreign attributes }>
  Content: (annotation?, (restriction | list | union))
</simpleType>

http://www.w3.org/TR/xmlschema-1/#declare-datatype
Derivation

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

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

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

http://www.w3.org/TR/xmlschema-1/#declare-datatype
```

Datentypen

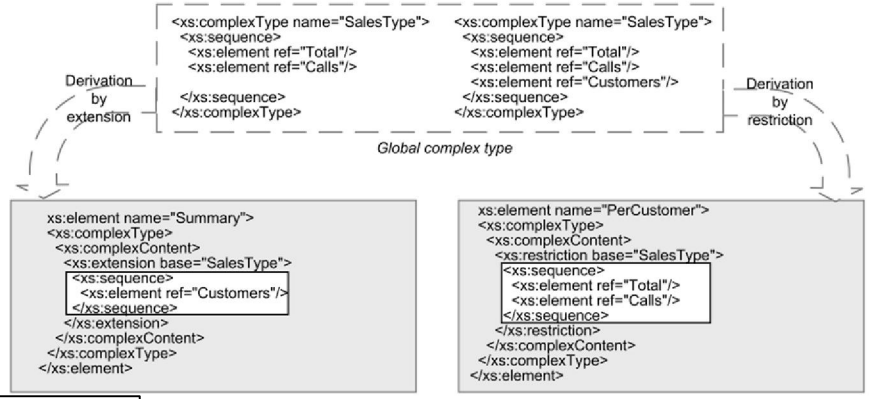



```
<?xml version="1.0" encoding="UTF-8" ?>
<bond_movies
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="bond_movies.xsd"
      month="August" year="2013">
  <movie number="_01">
    <title>Dr. No</title>
    <bond>Sean Connery</bond>
    <bond_girl>Ursula Andress</bond_girl>
    <year>1962</year>
    <duration>105</duration>
  </movie>
  <movie number="_02">
    <title>From russia with love</title>
    <bond>Sean Connery</bond>
    <bond_girl>Daniela Bianchi</bond_girl>
    <year>1963</year>
    <duration>110</duration>
  </movie>
</bond_movies>
```

Simple Element / Restriction

```
<xs:element name="name" type="xs:string"/>
<xs:element name="life_exp_women" type="xs:decimal"/>>
<xs:element name="population_under_15" type="percentType"/>
<xs:simpleType name="percentType">
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Derivative



Schema without Target Namespace

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!- Schema content -->
</xs:schema>
```

Schema with Target Namespace

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.demo.ch/countries"
           targetNamespace="http://www.demo.ch/countries"
           elementFormDefault="qualified">
<!- Schema content -->
</xs:schema>
```

```
<xs:simpleType name="beer">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Lozaerner Bier"/>
    <xs:enumeration value="Eichhof"/>
    <xs:enumeration value="Boxer"/>
  </xs:restriction>
</xs:simpleType>
```

Data-Centric: Die Daten stehen im Mittelpunkt, somit die Struktur wird priorisiert
Document-Centric: Der Text steht im Mittelpunkt, die Struktur von XML ist zweitrangig (z.B. XHTML)

Bei SOAP werden XML Daten ausgetauscht. → Der User arbeitet mit XML
Bei XML-RPC werden die Daten in XML„ verpackt“ und via Middleware transportiert .→ User merkt nichts davon

Keys and references

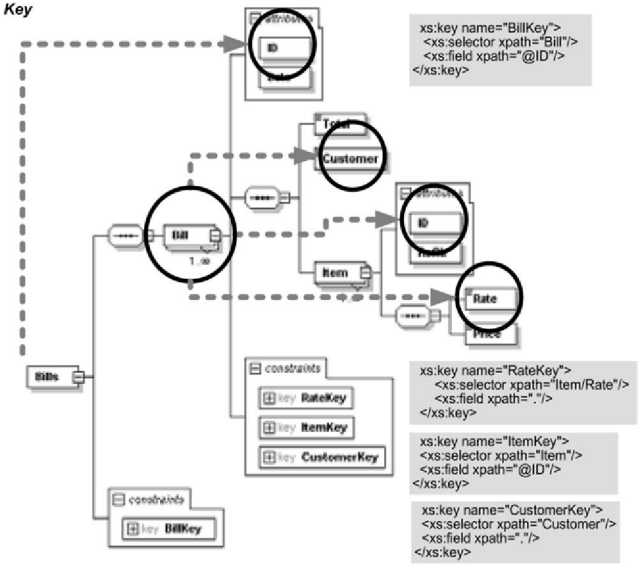
Definition

```
<unique
  id = ID
  name = NCName
  { foreign attributes }>
  Content: (annotation?, (selector, field+))
</unique>

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

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

Field reference

```
<selector
  id = ID
  xpath = Subset of  XPath
  { foreign attributes }>
  Content: (annotation?)
</selector>

<field
  id = ID
  xpath = Subset of XPath
  { foreign attributes }>
  Content: (annotation?)
</field>
```

Key




```
xs:key name="BillKey">
  <xs:selector xpath="Bill"/>
  <xs:field xpath="@ID"/>
</xs:key>
```

```
xs:key name="RateKey">
   <xs:selector xpath="Item/Rate"/>
   <xs:field xpath="."/>
</xs:key>
```

```
xs:key name="ItemKey">
<xs:selector xpath="Item"/>
<xs:field xpath="@ID"/>
</xs:key>
```

```
xs:key name="CustomerKey">
<xs:selector xpath="Customer"/>
<xs:field xpath="."/>
</xs:key>
```

Any

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

http://www.w3.org/TR/xmlschema-1/#declare-openness

Notation

```
<notation
  id = ID
  name = NCName
  public = anyURI
  system = anyURI
  { foreign attributes }>
  Content: (annotation?)
</notation>
```

Documentation

```
<annotation
  id = ID
  { foreign attributes }>
  Content: (appinfo | documenta-
tion)*
</annotation>

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

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

```
<xs:element name="Sales" type="xs:decimal"/>
<xs:element name="Calls" type="xs:decimal"/>
<xs:element name="Customers" type="xs:short"/>

Global Elements

<xs:complexType name="TotalType">
  <xs:sequence>
    <xs:element ref="Sales"/>
    <xs:element ref="Customers"/>
    <xs:element ref="Calls"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType
      name="PerCustomerType">
  <xs:sequence>
    <xs:element ref="Sales"/>
    <xs:element ref="Calls"/>
  </xs:sequence>
</xs:complexType>

Global complex types

<xs:complexType name="SuccessType">
  <xs:sequence>
    <xs:element name="Total" type="TotalType"/>
    <xs:element name="PerCustomer" type="PerCustomerType"/>
  </xs:sequence>
  <xs:attribute name="City" type="xs:string" use="required"/>
  <xs:attribute name="Month" type="xs:decimal" use="required"/>
</xs:complexType>

Root element

<xs:element name="SuccessSummary">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Success" type="SuccessType"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Content models

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

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

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

Overview

Primary components	Secondary components	Auxiliary components
1. Simple type definitions	5. Attribute group definitions	9. Model groups
2. Complex type definitions	6. Identity-constraint definitions	10. Particles
3. Attribute declarations	7. Model group definitions	11. Wildcards
4. Element declarations	8. Notation declarations	12. Attribute Uses
		13. Annotations

Facets

Facet	Description
length	String length
minLength	Minimum string length
maxLength	Maximum string length
pattern	Regular expression
enumeration	Enumeration of allowed values
whiteSpace	Whitespace treatment
maxInclusive	Upper inclusive limit
maxExclusive	Upper exclusive limit
minExclusive	Lower exclusive limit
minInclusive	Lower inclusive limit
totalDigits	Total digits of a (decimal) number
fractionDigits	Fraction digits of a (decimal) number

Complex Type

- <xs:sequence> elements must appear in the given order
- <xs:choice> if exactly one element is to be selected
- <xs:all> elements can appear zero or one time in any order

```
<xs:element name="pets">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="dog" type="xs:string"/>
      <xs:element name="cat" type="xs:string"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="xs:string"/>
      <xs:element name="player" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Combining Complex Types

```
<xs:complexType name="NameOrEmail">
  <xs:choice>
    <xs:element name="email" type="xs:string"/>
    <xs:sequence>
      <xs:element name="first" type="xs:string"/>
      <xs:element name="last" type="xs:string"/>
    </xs:sequence>
  </xs:choice>
</xs:complexType>
```

Simple Elements with Attributs / Extension

```
<product price="2.00">Coffee in HSLU canteen</product>

<xs:element name="product">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="price" type="xs:decimal" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Schema Features: Mixed Content

```
<abstract>Hello, <b>I</b> am <i>a</i> silly test with way too much
<a>formatting</a>.</abstract>

<xs:element name="abstract">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="i" type="xs:string"/>
      <xs:element name="b" type="xs:string"/>
      <xs:element name="a" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Schema Features: Wildcards

```
<xs:sequence>
  <xs:element name="Title" type="xs:string"/>
  <xs:element name="Author" type="xs:string"/>
  <xs:any namespace="##any" minOccurs="0"/>
</xs:sequence>
```

After <Author> one other well-formed XML element from any namespace may come.

Schema Features: Unique ID

```
<xs:unique name="movieID">
  <xs:selector xpath="movie"/>
  <xs:field xpath="@number"/>
</xs:unique>
 <!-- oder -->
<xs:key name="movieID">
  <xs:selector xpath="movie"/>
  <xs:field xpath="@number"/>
</xs:key>
```

Schema Features: Key References

```
<xs:element name="bibliography" type="bibliographyType">
  <xs:key name="bKey">
    <xs:selector xpath="*"/>
    <xs:field xpath="@key"/>
  </xs:key>
  <xs:keyref name="bKeyRef" refer="bKey">
    <xs:selector xpath="*/cite"/>
    <xs:field xpath="@item"/>
  </xs:keyref>
</xs:element>
```

XSLT – QuickReference

Basic elements

Root

```
<xsl:stylesheet
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number
  xpath-default-namespace = uri
  default-validation = "strict" | "lax" | "preserve" | "strip">
  <!-- Inhalt: (xsl:import*, other-declarations) -->
</xsl:stylesheet>
```

Output

```
<xsl:output
  name = qname
  method = "xml" | "html" | "xhtml" | "text" | qname-but-not-ncname
  cdata-section-elements = qnames
  doctype-public = string
  doctype-system = string
  encoding = string
  escape-uri-attributes = "yes" | "no"
  include-content-type = "yes" | "no"
  indent = "yes" | "no"
  media-type = string
  normalize-unicode = "yes" | "no"
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  undeclare-namespaces = "yes" | "no"
  use-character-maps = qnames
  version = nmtoken />
```

Whitespace

Deletes whitespace from the elements in the source document which are named in the token list.

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

Preserves whitespace in the listed elements.

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

Push vs. Pull
<for-each> pull-processing
Templates: push-processing

Templates
-Templates are more flexibel and easier to maintain
-Templates can be shared between stylesheets
-Templates can be overwritten
-Templates enable modular code

Legend

[element1| element2]
Either element1 or element2 from the element group
[]*
Repeated use of group possible
{ }
Optional declaration

Templates

Definition

Defines a template which can be addressed by its name in the name attribute or which is automatically selected by the XPath pattern in the match attribute. An additional name in the mode attribute can be used to create templates with the same name but different modes i.e. different processing under the same name or for the same pattern.

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

Applying templates

Applies suitable templates which are selected by the XPath patterns in the select attribute and – if existing – the value in the mode-attribute.

```
<xsl:apply-templates
  select = node-sequence-Ausdruck
  mode = token>
  <!-- Inhalt: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

Calls a template by using the qualified name and applies it to the selected node list.

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

Use of the imported templates instead of the already selected template.

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

Additional use of the imported templates following the priorities in the import tree.

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

Output

Outputs the PCDATA text in the result document with – if necessary – parsing of internal entities.

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

Outputs the result of the XPath pattern (e.g. the content of a simple text node or the result of a calculation).

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

Choose - Statement

```
<xsl:template match="movie">
  <xsl:choose>
    <xsl:when test="duration &gt; 140">
      <li style="color:red"><xsl:value-of select="title"/></li>
    </xsl:when>

    <xsl:when test="duration &gt; 130">
      <li style="color:blue"><xsl:value-of select="title"/></li>
    </xsl:when>

    <xsl:when test="duration &gt; 120">
      <li style="color:orange"><xsl:value-of select="title"/></li>
    </xsl:when>

    <xsl:otherwise>
      <li style="color:green"><xsl:value-of select="title"/></li>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

XML Output

Elements

Creates an XML element with the name specified in the name attribute.

```
<xsl:element
  name = { qname }
  namespace = { uri-reference }
  use-attribute-sets = qnames
  validation = "strict" | "lax" | "preserve" | "strip"
  type = qname>
  <!-- Inhalt: sequence-constructor -->
</xsl:element>
```

Comments

Creates an XML comment containing the text node as ouput.

```
<xsl:comment>
  <!-- Inhalt: sequence-constructor -->
</xsl:comment>
```

Attributes

Creates an XML attribute with the name specified in the name attribute.

```
<xsl:attribute
  name = { qname }
  namespace = { uri-reference }
  validation = "strict" | "lax" | "preserve" | "strip"
  type = qname
  disable-output-escaping = "yes" | "no">
  <!-- Inhalt: sequence-constructor -->
</xsl:attribute>
```

Copies attributes which are stored in global attribute sets into the element using a whitespace separated list of attribute set names.

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

Control Statements

Conditional processing

Encloses a conditional statement consisting of one or more when statements (cases) and additionally an optional default case which is defined in the otherwise element.

```
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
  <xsl:when
    test = expression>
    <!-- Content: sequence-constructor -->
  </xsl:when>
  <xsl:otherwise>
    <!-- Content: sequence-constructor -->
  </xsl:otherwise>
</xsl:choose>
```

Executes statements if the condition in the test attribute evaluates to true.

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

Loops

Creates an iteration for all nodes which are selected by the XPath expressions in the select attribute and executes the inner statements.

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

For-Each

```
<xsl:for-each select="bond_movies/movie">
  <li>
    <xsl:value-of select="title"/>
  </li>
</xsl:for-each>
```

IF - Statement

There is an IF-statement in XSLT but without ELSE:

```
<xsl:template match="movie">
  <xsl:if test="duration &gt; 130">
    <li style="color:red">
      <xsl:value-of select="title"/>
    </li>
  </xsl:if>

  <xsl:if test="duration &lt;= 130">
    <li>
      <xsl:value-of select="title"/>
    </li>
  </xsl:if>
</xsl:template>>
```

Dynamic Values

Variables

Creates a variable with the name being defined in the name attribute. The value is either defined in the select attribute or by the executed statements in the sequence constructor. The scope of the variable is either the element in which it was declared or the stylesheet that is declared as a child element of xsl:stylesheet.

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

Parameters

Creates a parameter for a template (if declared within xsl:template) or the stylesheet (if directly declared within xsl:stylesheet). A default value can be stored in the select attribute or by the execution of the sequence constructor.

```
<xsl:param
  name = qname
  select = Ausdruck
  as = sequence-type
  required = "yes" | "no">
  <!-- Inhalt: sequence-constructor -->
</xsl:param>
```

Sets the value of a parameter when using xsl:apply templates or xsl:call template. The value is either stored in the select attribute or generated by the execution of the statements in the sequence constructor.

```
<xsl:with-param
  name = qname
  select = Ausdruck>
  <!-- Inhalt: sequence-constructor -->
</xsl:with-param>
```

Copying

Copies the name of the context node into the result document.

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

Copies the selected tree into the result document.

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

Grouping

Groups the selected node set by the expression in the group-by attribute. Advanced grouping via start and end expressions is possible by using the attributes group-starting-with and group-ending-with. The function current-grouping-key() gets the present value of the grouping key which is evaluated by group-by. The function current-group() points to the context group and is used within the select attributes of xsl:apply templates or xsl:for-each.

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

Sorting

Sorts the selected nodes within xsl:apply templates and xsl:for-each.

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

Regular Expressions

Tests the regular expression in the regex attribute within the scope of the selected nodes.

```
<xsl:analyze-string
  select = Ausdruck
  regex = { string }
  flags = { string }>
  <!-- Content: (xsl:matching-substring?,
xsl:non-matching-substring?, xsl:fallback*) -->
```

Contains the statements for the matching nodes.

```
<xsl:matching-substring>
<!-- Content: sequence-constructor -->
</xsl:matching-substring>
```

Contains the statements for the non-matching nodes.

```
<xsl:non-matching-substring>
  <!-- Content: sequence-constructor -->
</xsl:non-matching-substring>
```

Modular XSLT files

Including files

External templates can be included before and after internal xsl:template elements. Therefore, external templates can override internal ones or be overridden by the internal templates depending on the position of the xsl:include element.

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

Importing files

The import statement has to be located before all internal xsl:template elements. Therefore, imported templates can only be overridden and they cannot override internal ones.

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

XML Schema

Imports an XML Schema document so that names of data types and global elements defined within the XML Schema can be referenced from the XSLT document.

```
<xsl:import-schema
  namespace = { uri-reference }
  schema-location = { uri-reference } />
```

Functions

Declares a user-defined function which can be used in XPath expression (select or test attributes).

```
<xsl:function
  name = qname
  as = sequence-type
  override = „yes" | „no">
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:function>
```

Keys

Generates a unique key which can be referenced from the key() function in XPath expressions.

```
<xsl:key
  name = qname
  match = pattern
  use = Ausdruck
  as = qname
  collation = { uri } />
```

Getting Content from other XML Files

In bond_movies.xml each movie has an attribute id.
Imagine now another file bond_movies_media.xml that stores poster images for each movie referenced by ID.

```
<img src="{document('bond_movies_media.xml')/bond_movies/movie[@number='_02']/poster/@href}"
  alt="{document('bond_movies_media.xml')/bond_movies/movie[@number='_02']/title}" />
```

Named Templates

```
<xsl:template name="header">
  <xsl:param name="color" />
  <tr style="background-color:{$color}">
    <th>Title</th>
    <th>Bond Actor</th>
    <th>Bond Girl</th>
    <th>Producer</th>
    <th>Year</th>
    <th>Length</th>
  </tr>
</xsl:template>
```

Calling Templates with Parameters

```
<xsl:call-template name="header">
<xsl:with-param name="color">#990066</xsl:with-param>
</xsl:call-template>
```

Sorting Elements

```
<xsl:template match="bond_movie">
…
<xsl:apply-templates
       select="movie[starts-with(bond/text(), 'Pierce')]">
  <xsl:sort select="bond_girl" data-type="text" order="descending"/>
</xsl:apply-templates>
…
</xsl:template>
```

Built-In Template

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

Variables

```
<xsl:variable name="version">1.0beta</xsl:variable>
```

However, variables can be initialized only once in their lifetime.

Copy-Of

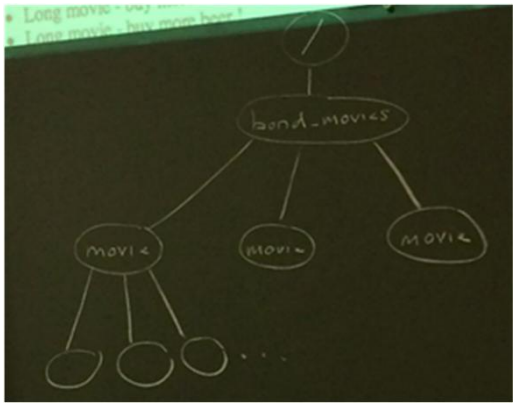
```
<copy-of select="path" />
```

copies content from the source to the output tree, i.e. it copies the specified node with its children and attributes. This is very handy if you want to transform a document into a modified form of itself.

Import

```
<import href="path" />
```

includes another stylesheet by just copy-pasting its content to the current file. This enables to build modules of reusable* code. In case of conflicts, the imported templates obtain lower priority.



XSLT in a Browser

```
<?xml-stylesheet type="text/xsl" href="bond_movie_list.xsl"?>
```

Template Context

Der Context ist dasjenige Element worauf das Template aufgerufen wird. Der Context ist eine Liste von Knoten, welches aktuell vom Template bearbeitet wird.
Innerhalb vom Context kann nicht „zurück" gesprungen werden, es kann nun nach „vorne" gehen.

XSLT Constraints

Input muss XML sein (weil Matches mit XPath definiert sind)
Output kann (muss aber nicht) XML sein.

Conflict Resolution

Import: import hat die kleinere Priorität
Template: dasjenige welches am besten zutrifft, falls zwei gleich sind wird das zuletzt definierte verwendet.
Built-In Template kommen zum Einsatz wenn sonst keines gefunden wird.

Schema Bond Collection

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="bond_movies">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="actor" minOccurs="1" maxOccurs="unbounded"/>
        <xs:element ref="movie" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="month" type="monthType" use="required"/>
      <xs:attribute name="year" type="xs:gYear" use="required"/>
    </xs:complexType>

    <xs:key name="actorID">
      <xs:selector xpath="actor"/>
      <xs:field xpath="@id"/>
    </xs:key>

    <xs:keyref name="actorRef" refer="actorID">
      <xs:selector xpath="movie"/>
      <xs:field xpath="@actor"/>
    </xs:keyref>
  </xs:element>

  <xs:element name="actor">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="id" type="xs:integer"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="movie">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="bond_girl" type="xs:string"/>
        <xs:element name="regie" type="xs:string"/>
        <xs:element name="year" type="xs:gYear"/>
        <xs:element name="duration" type="durationType"/>
      </xs:sequence>
      <xs:attribute name="actor" type="xs:integer"/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="durationType">
    <xs:restriction base="xs:short">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="300"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="monthType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="January"/>
      <xs:enumeration value="February"/>
      <xs:enumeration value="March"/>
      <xs:enumeration value="May"/>
      <xs:enumeration value="June"/>
      <xs:enumeration value="July"/>
      <xs:enumeration value="August"/>
      <xs:enumeration value="September"/>
      <xs:enumeration value="October"/>
      <xs:enumeration value="November"/>
      <xs:enumeration value="December"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="numberType">
    <xs:restriction base="xs:string">
      <xs:length value="3"/>
      <xs:pattern value="_\d{2}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Push / Pull (Templates)

<for-each> pull-processing
Templates: push-processing

-Templates are more flexibel and easier to maintain
-Templates can be shared between stylesheets
-Templates can be overwritten
-Templates enable modular code

XML Bond Collection

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="bond_movie_list.xsl"?>
<bond_movies month="August" year="2013">
  <movie number="_01">
    <title>Dr. No</title>
    <bond>Sean Connery</bond>
    <bond_girl>Ursula Andress</bond_girl>
    <regie>Terence Young</regie>
    <year>1962</year>
    <duration>105</duration>
  </movie>
  <movie number="_02">
    <title>From russia with love</title>
    <bond>Sean Connery</bond>
    <bond_girl>Daniela Bianchi</bond_girl>
    <regie>Terence Young</regie>
    <year>1963</year>
    <duration>110</duration>
  </movie>
  <movie number="_03">
    <title>Goldfinger</title>
    <bond>Sean Connery</bond>
    <bond_girl>Honor Blackman</bond_girl>
    <regie>Guy Hamilton</regie>
    <year>1964</year>
    <duration>106</duration>
  </movie>
  <movie number="_04">
    <title>Thunderball</title>
    <bond>Sean Connery</bond>
    <bond_girl>Claudine Auger</bond_girl>
    <regie>Terence Young</regie>
    <year>1965</year>
    <duration>125</duration>
  </movie>
</bond_movies>
```

XSL Bond Collection

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
 <xsl:output doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />
  <xsl:template match="/">
    <html>
    <body>
      <h1>James Bond Movies</h1>
      <table border="1">
        <tr>
          <th>Title</th>
          <th>Actor</th>
          <th>Duration</th>
        </tr>
        <xsl:apply-templates select="bond_movies/movie" />
      </table>
    </body>
    </html>
  </xsl:template>

  <xsl:template match="movie">
    <xsl:if test="((position() mod 2) = 0)">
      <tr bgcolor="yellow">
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="bond"/></td>
        <td><xsl:value-of select="duration"/></td>
      </tr>
    </xsl:if>

    <xsl:if test="((position() mod 2) != 0)">
      <tr bgcolor="lightgreen">
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="bond"/></td>
        <td><xsl:value-of select="duration"/></td>
      </tr>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

SAX | Simple API for XML

- Push Prinzip
- Überschreibt einzelne Events
- Resourceneffizient
- Kompliziert
- sequenziell
- Kein Weg zurück

SAX Event List

```
<?xml version="1.0"?>
<doc>
  <para>Hello world!</para>
</doc>
```

start document
start element (doc)
start element (para)
characters (Hello world!)
end element (para)
end element (doc)
end document

SAX Handler Class / Main

1. Create class extending DefaultHandler
2. Override listener methods

```
public class BondHandler extends DefaultHandler {
  private String current;
  @Override
  public void startElement(String uri, String local, String name,
    Attributes attr) {
    current = name;
  }
  @Override
  public void characters(char[] ch, int start, int length) {
    if (current != null && current.equals("title")) {
      System.out.println(new String(ch, start, length));
      current = null;
} } }
// MAIN
```

1. Create a parser (XMLReader)
2. Give DefaultHandler extension to parser
3. Start reader on XML file

```
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setContentHandler(new BondHandler());
File file = new File("../bond_movies.xml");
reader.parse(new InputSource(new FileReader(file)));
```

DOM / JDOM

- In Memory Tree-Darstellung der XML Datei
- Memorylastig
- DOM != JDOM, nur Grundidee ist gleich
- JDOM verwendet Java-Typen (optimiert)
- JDOM Parser sind SAXBuilder und DOMBuilder

1. SAXBuilder returns JDOM data structure
2. Access the root node
3. Browse the tree

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(new FileInputStream("../bond_movies.xml"));
Element root = doc.getRootElement();
for(Element e : root.getChildren()) {
  System.out.println(e.getChildText("title"));
}
```

XSLT from Java | XSLT Transform

1. Create transformer with
1.1 a streamsource for .xsl file
1.2 a streamsource for input XML file
1.3 a streamresult for output XML file
2. Start transformation

```
TransformerFactory factory = TransformerFactory.newInstance();
Transformer transformer = factory.newTransformer(
     new StreamSource("../../XML 5 - XSLT/bond_movie_xhtml_table.xsl"));
transformer.transform(
  new StreamSource("../bond_movies.xml"),
  new StreamResult("../result.xhtml"));
```

XML Schema Validation with JAXP

1. Specify the validation method
2. Create schema from file
3. Produce validator from schema
4. Apply validator to XML file

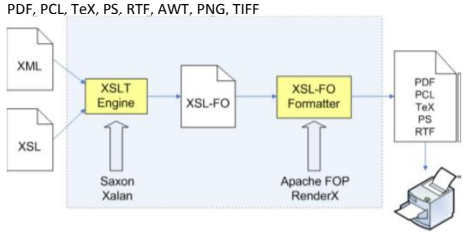
```
SchemaFactory factory =
          SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Source schemaSource = new StreamSource(new File("bond_movies.xsd"));
Schema schema = factory.newSchema(schemaSource);
Validator validator = schema.newValidator();
validator.validate(new StreamSource("bond_movies.xml"));
```

Unmarshalling XML with JAXB

```
JAXBContext jc = JAXBContext.newInstance("jaxb.bond");
Unmarshaller unmarshaller = jc.createUnmarshaller();
BondMovies movies =
    (BondMovies) unmarshaller.unmarshal(new File("../bond_movies.xml"));
for(Movie m : movies.getMovie()) {
  System.out.println(m.getTitle());
}
```

Formatting Objects (XSL-FO)

PDF, PCL, TeX, PS, RTF, AWT, PNG, TIFF



2-Phasen:

XML -> (XSL) -> FO -> (FOP) -> PDF
[Inhalt] [Layout] [Format]
Die erste Phase trennt Inhalt von Layout, die zweite Phase trennt das Format vom Layout.

FO-Document:

Wurzelknoten, zwei Sektionen: Layout / Content-Section

```
<xsl:template match="/">
  <fo:root>
    <fo:layout-master-set>    <!-- layout section  -->
      ...
    </fo:layout-master-set>
    <fo:page-sequence>        <!-- content section -->
      ...
    </fo:page-sequence>
  </fo:root>
</xsl:template>
```

Scalable Vector Graphics | Bitmap / Vektor GFX

Bitmap: Matrix von Farbwerte
Vektor: Geometrische Beschreibung

SVG Path

Ein Path sind Punkte die mit Linien verbunden werden. Die Linien können gerade oder kurven sein.
Die Punkte können absolut (upper-case) oder relativ (lower-case) definiert werden (vom Ursprung aus)

XLink

Grundsätzlich zwei Objekte miteinander verbinden. Umgesetzt wurden nur die „simple links“.
In HTML die „href“ bsp. beim Link … oder bei SVG um Pfade mit dem Objekt zu verbinden.
→ Trennen von Definition und Verwendung

Scalable Vector Graphics | Animation → SMIL

Animation is the time-based manipulation of an attribute of a target element.

Xlink & SVG

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg:svg xmlns:svg="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <svg:defs>
  <svg:path id="path1" d="M 100 200 C 200 100 300 0 400 100
       C 500 200 600 300 700 200 C 800 100 900 100 900 100" />
  </svg:defs>

  <svg:use xlink:href="#path1" fill="none" stroke="red" />

    <svg:text font-family="Verdana" font-size="42.5">
    <svg:textPath xlink:href="#path1">
      We go up, then we go down, then up again
    </svg:textPath>
  </svg:text>
</svg:svg>
```

XHTML Quick Reference

Rules

Declare a DOCTYPE

For a list, see http://www.w3.org/QA/2002/04/valid-dtd-list.html

```
<!DOCTYPE doctype goes here >
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>page title</title></head>
<body>content ...</body>
</html>
```

Write tags in all lowercase

✗ <TABLE>
✗ <Table>
✓ <table>

Close all tags

✗
✓
✗ <img src="pix.gif">
✓

Nest tags in order

✗ text
✓ text

Use quotes for attribute values

✗ <table width=50%>
✓ <table width="50%">

Use id instead of name

The name attribute has been replaced with id.

✗
✓

Hide scripts with CDATA

JavaScript code should be enclosed in CDATA ("character data") sections.

```
<script type="text/javascript">
/* <![CDATA[ */
javascript code
/* ]]> */
</script>
```

Tags

Structure

<h1>	Heading 1 (through <h6>)
<p>	Paragraph
 	Line break
	Block of content
	Inline block of content

Formatting

<hr />	Horizontal rule/line
	Emphasis (italic)
	Bold
<sub>	Subscript (H_2O)
<sup>	Superscript text ($E=mc^2$)

Links

Hyperlink	*link text*
Bookmark	*link text*
Email	*link text*
Stylesheet	<link rel="stylesheet" href="styles.css" type="text/css" />
JavaScript	<script language="javascript" type="text/javascript" src="code.js" />

Special characters

Non-breaking space		Double quotes (")	"
Ampersand (&)	&	Copyright (©)	©
Greater than sign (>)	>	Trademark (™)	™
Less than sign (<)	<	Registered trademark (®)	®

Forms

```
<form>
  <fieldset>
  <input type="text" />
  <input type "checkbox" />
  <input type="radio /">
  <select />
  <input type="submit" />
  </fieldset>
</form>
```

Submit

Tables

```
<table>
<tr><th>head</th><th>head</th></tr>
<tr><td>cell</th><td>cell</th></tr>
</table>
```

Bulleted lists

```
<ul>
  • <li>...</li>
  • <li>...</li>
</ul>
```

Numbered lists

```
<ol>
  1. <li>...</li>
  2. <li>...</li>
</ol>
```

Relative Location Paths

Relative Location Paths traverse the document from the context node

para
para element children
Also – **child::para**

@type
the **type** attribute
Also – **attribute::type**

../title
the **title** element children of the parent

*** except title**
child elements except **title** elements
Also – ***[not(self::title)]** (works in XPath 1.0)

ancestor::sec
all **sec** ancestor elements

ancestor::sec/@n
all **n** attributes on **sec** ancestor elements

list/(item | step)
item and **step** element children of **list** children, in document order

list/item, list/step
item element children of **list** children followed by **step** children of **list** children

preceding-sibling::step
all preceding sibling **step** elements

preceding-sibling::*[1][self::step]
the directly preceding sibling element, if it is a **step** (otherwise nothing)

descendant::div[last()]
the last **div** descendant of the current node

.//div[last()]
div descendants that are the last child **div** of each of their parents

preceding::pb[1]
the first (most immediate) preceding **pb**

ancestor::sec//pb intersect preceding::pb
pb elements inside the same **sec** element as the context node, preceding it

p[normalize-space()]
p child elements that have a non-whitespace value (text content)

***[not(node())]**
empty element children (i.e., element children with no node children)

***[not(node() except (comment()|processing-instruction()))]**
element children that are empty (have no children) except for comments or processing instructions

step[position() gt 1]
all **step** element children but the first

step except *[1]
step element children but the first

step[position() le 4]
the first four **step** element children
Also – **step[position() = (1 to 4)]**

step[position() mod 2]
odd-numbered **step** children

step[not(position() mod 2)]
even-numbered **step** children

***[position() le 4] intersect step**
from the first four element children, the **step** children

ancestor-or-self::*[exists(@lang)][1]/@lang
the closest **lang** attribute on the context node or an ancestor element

Expressions that are not Location Paths

(@class,'none')[1]
the **class** attribute, or if it does not exist, the string **"none"**.
Also –
if (exists(@class)) then @class else "none"

//*/name()
the names of all elements, in document order

distinct-values(//*/name())
the names of all elements, in document order, with duplicates removed

//name/string-join((first, last),' ')
a sequence of strings constructed from the **name** elements in the document, each one concatenating the values of its **first** and **last** element children, in that order, joining them with spaces
Also – **for $n in //name return string-join(($n/first,$n/last),' ')**

//*/count(ancestor-or-self::*)
a sequence of numbers representing the depth of each element in the document

max(//*/count(ancestor-or-self::*))
the maximum depth of all elements in the document (a number in a singleton sequence)

for $stooge in ('Moe','Larry','Curly') return count(//p[contains(.,$stooge)])
the counts of all **p** elements in the document mentioning each of "Moe", "Larry" and "Curly", in that order

index-of(('Moe','Larry','Curly'), speaker[1])
if the first **speaker** element child has the value "Moe", then 1; if "Larry", then 2; if "Curly", then 3; otherwise the empty sequence (i.e., no value)

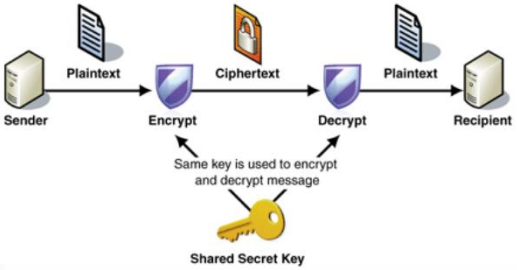
(: You've got to be kidding me. :)
do nothing. A comment is just a comment.

2008-07-21

XPath 2.0 Quick Reference

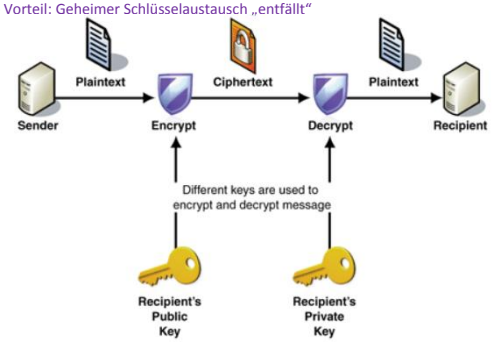
Symmetric Encryption

Sender und Empfänger benützen den gleichen Key
- Nachteil: Schlüsselaustausch muss „sicher" sein
- Vorteil: Sehr Effizient



Asymmetric Encryption

Sender verschlüsselt mit dem Public Key des Empfängers. Der Empfänger entschlüsselt es mit seinem Private-Key.
- Nachteil: Aufwendig zum berechnen
- Vorteil: Geheimer Schlüsselaustausch „entfällt"



Hybrid Cryptography

Schlüsselaustausch wird asymmetrisch übertragen.
Die Nachrichten werden nachher symmetrisch übermittelt.

XML Encryption Granularities

- Gesamtes XML-Dokument
- Einzelnes Element in einem XML-Dokument
- Einzelner Inhalt von einem Element in einem XML-Dokument

Digitale Signatur

- Authentifiziert der Absender
- Die Integrität der Nachricht kann überprüft werden
- Die „nicht ab Streitbarkeit" Bsp. Verträge: Der Absender kann nachträglich die Nachricht nicht „leugnen"

Inhalt der XML Signatur

- verschlüsselte Nachricht
- Verschlüsselung Algorithmus
- Verwendete Schlüssel (Key)
- Kanonisierungsmethode

Absolute Location Paths

Absolute Location Paths traverse the document starting at the top (the root), and can be recognized by their initial / (forwardslash).

/book/bookinfo/abstract
an **abstract** element child of a **bookinfo** child of the **book** document element
Also –
/child::book/child::bookinfo/child::abstract

//para
all **para** elements in the document
Also – **/descendant-or-self::*/child::para**
Also – **/descendant::para**

/descendant::para[1]
the first **para** element in the document
Also – **(//para)[1]**

//@order-by
all **order-by** attributes in the document

//list[exists(ancestor::list)]
all **list** elements that have ancestor **list**elements

//list[not(ancestor::list)]
all **list** elements that do not have ancestor **list** elements
Also – **//list[not(exists(ancestor::list))]**
Also – **//list[empty(ancestor::list)]**

//(* except title)
all elements except **title** elements
Also – **//*[not(self::title)]** (works in XPath 1.0)

//processing-instruction()[not(ancestor::sec/@n = 1)]
all processing instructions with no **sec** ancestor elements with **n** attributes equal to 1

//para[matches(.,'[X|x]{3}')]
all **para** elements whose value includes the regular expression **[X|x]{3}**
Tip – **[X|x]{3}** matches three **X** or **x**characters appearing in a row

//sec[@id = //@rid/tokenize(.,'\s+')]
all **sec** elements with **id** attributes whose values are also given as a value by a tokenized **rid** attribute anywhere in the document
Also – **//sec[@id = $rid-values]** where **$rid-values** is **distinct-values(//@rid/tokenize(.,'\s+'))**
Tip – use **distinct-values(//@rid/tokenize(.,'\s+'))** to remove duplicates from the list of tokenized **@rid** values
Tip – the regular expression **\s+** matches any contiguous sequence of spaces (space, linefeed or tab characters)

Simple Expressions

$VarName
(Expr)
( )
. (one dot: self)
QName (Expr , ...)
QName ( )
IntegerLiteral
DecimalLiteral
DoubleLiteral
StringLiteral

Arithmetic Expressions

+ Expr	Expr + Expr
- Expr	Expr - Expr
Expr * Expr	Expr **div** Expr
Expr **idiv** Expr	Expr **mod** Expr

Creating Sequences

Create a sequence from a list of items:
Expr , ...
Note: A sequence list must usually be parenthesized.

Repeat over one or more sequences, returning a sequence of results:
for VariableBinding , ... **return** Expr

where a VariableBinding is:
$VarName **in** Expr

Create a numeric sequences, from lower bound to upper bound:
Expr **to** Expr

All the items appearing in either sequence:
Expr **union** Expr
Expr | Expr

Only items appearing in both sequences:
Expr **intersect** Expr

All items in the first sequence not in second:
Expr **except** Expr

Comments in XPath Expressions

(: This is a comment within an XPath expr :)

Testing

Test if the condition is satisfied for at least one combination of the bound expressions:
some VariableBinding , ... **satisfies** Expr

Test if the condition is satisfied for all of the bound expressions:
every VariableBinding , ... **satisfies** Expr

Select one or the other of two possibiliites:
if (Expr) **then** Expr **else** Expr

Either or both of two tests:
Expr **or** Expr Expr **and** Expr

Test if they are the same node:
Expr **is** Expr

Test if a node appears before or after another:
Expr << Expr Expr >> Expr

Test an expression's dynamic type:
Expr **instance of** SequenceType

Test if an expression can be converted to a type:
Expr **castable as** AtomicType
Expr **castable as** AtomicType?

Compare two atomic values:

Expr **eq** Expr	Expr **ne** Expr
Expr **lt** Expr	Expr **le** Expr
Expr **gt** Expr	Expr **ge** Expr

Compare all items in one sequence to all items in a second, and return if true for any pair of values:

Expr = Expr	Expr != Expr
Expr < Expr	Expr <= Expr
Expr > Expr	Expr >= Expr

Type Modification Expressions

Use as without converting:
Expr **treat as** SequenceType

Use as, converting as needed and doable:
Expr **cast as** AtomicType
Expr **cast as** AtomicType?

XPath 2.0:
http://www.w3.org/TR/xpath20/
XSL-List:
http://www.mulberrytech.com/xsl/xsl-list

Path Expressions

/	Top level, document root
/ Step	At top level
Step	Relative to current node
// Step	Anywhere within document
Path / Step	Immediately within Path
Path // Step	Anywhere within Path

Where a Step is one of:
Expr
AxisName::NameTest
AxisName::KindTest

@NameTest	(attribute test)
NameTest	(child element test)
KindTest	(child node test)
..	(two dots: parent test)

Followed by zero or more predicates:
[Expr]

Where an AxisName is one of:

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

Where a NameTest is one of:
QName
*
NCName:*
*:NCName

Where a KindTest is one of:
attribute (AttributeName)
attribute (AttributeName , TypeName)
attribute (*)
attribute (* , TypeName)
attribute ( )
comment ( )
document-node (**element** ...)
document-node (**schema-element** ...)
document-node ( )
element (ElementName)
element (ElementName , TypeName)
element (*)
element (* , TypeName)
element ( )
node ( )
processing-instruction (NCName)
processing-instruction (StringLiteral)
processing-instruction ( )
schema-attribute (AttributeName)
schema-element (ElementName)
text ( )

Names and Types

XML QNames, with or without a colon-separated prefix, is use for all of:
VarName
AttributeName
ElementName
TypeName
AtomicType

A SequenceType is one of:
empty-sequence ( )
KindTest
item ( )
AtomicType

Where KindTest, **item()** or AtomicType can be optionally followed by:

?	(may be empty sequence)\
+	(is a non-empty sequence of the type)
*	(is a sequence of the type, empty or not)

Operator Precedence:

1	, (comma)
2	**for** **some** **every** **if**
3	**or**
4	**and**
5	= != < <= > >= **eq** **ne** **lt** **le** **gt** **ge** **is** << >>
6	**to**
7	(two-argument) + -
8	* **div** **idiv** **mod**
9	**union** |
10	**intersect** **except**
11	**instance of**
12	**treat as**
13	**castable as**
14	**cast as**
15	(one-argument) + -
16	/ //
17	step node-test $name (Expr) function-call literal

Date/Time Functions

adjust-date-to-timezone(xs:date?) **as** xs:date?
adjust-date-to-timezone(xs:date?, xs:dayTimeDuration?) **as** xs:date?
adjust-dateTime-to-timezone(xs:dateTime?) **as** xs:dateTime?
adjust-dateTime-to-timezone(xs:dateTime?, xs:dayTimeDuration?) **as** xs:dateTime?
adjust-time-to-timezone(xs:time?) **as** xs:time?
adjust-time-to-timezone(xs:time?, xs:dayTimeDuration?) **as** xs:time?
dateTime(xs:date?, xs:time?) **as** xs:dateTime?
day-from-date(xs:date?) **as** xs:integer?
day-from-dateTime(xs:dateTime?) **as** xs:integer?
days-from-duration(xs:duration?) **as** xs:integer?
hours-from-dateTime(xs:dateTime?) **as** xs:integer?
hours-from-duration(xs:duration?) **as** xs:integer?
hours-from-time(xs:time?) **as** xs:integer?
implicit-timezone() **as** xs:dayTimeDuration
minutes-from-dateTime(xs:dateTime?) **as** xs:integer?
minutes-from-duration(xs:duration?) **as** xs:integer?
minutes-from-time(xs:time?) **as** xs:integer?
month-from-date(xs:date?) **as** xs:integer?
month-from-dateTime(xs:dateTime?) **as** xs:integer?
months-from-duration(xs:duration?) **as** xs:integer?
seconds-from-dateTime(xs:dateTime?) **as** xs:decimal?
seconds-from-duration(xs:duration?) **as** xs:decimal?
seconds-from-time(xs:time?) **as** xs:decimal?
timezone-from-date(xs:date?) **as** xs:dayTimeDuration?
timezone-from-dateTime(xs:dateTime?) **as** xs:dayTimeDuration?
timezone-from-time(xs:time?) **as** xs:dayTimeDuration?
year-from-date(xs:date?) **as** xs:integer?
year-from-dateTime(xs:dateTime?) **as** xs:integer?
years-from-duration(xs:duration?) **as** xs:integer?

Geben Sie die Namen aller Studenten aus, die eine Vorlesung bei „Russel“ hören.

```
<Studenten>
{
  for $n in distinct-values (
    for $v in //Vorlesung ,
        $s in //Student
    where contains($s/@hoert, $v/@VorlNr)
          and $v/../../Name = "Russel"
    return
      $s/Name
    )
  return <Name>{$n}</Name>
}
</Studenten>
```

XSLT-Only Functions

current() **as** item()
current-group() as item()*
current-grouping-key() **as** xs:anyAtomicType?
document(item()*) **as** node()*
document(item()*, node()) **as** node()*
element-available(xs:string) **as** xs:boolean
format-dateTime(xs:dateTime?, xs:string, xs:string?, xs:string?, xs:string?) **as** xs:string?
format-dateTime(xs:dateTime?, xs:string) **as** xs:string?
format-date(xs:date?, xs:string, xs:string?, xs:string?, xs:string?) **as** xs:string?
format-date(xs:date?, xs:string) **as** xs:string?
format-number(numeric?, xs:string) **as** xs:string
format-number(numeric?, xs:string, xs:string) **as** xs:string
format-time(xs:time?, xs:string, xs:string?, xs:string?, xs:string?) **as** xs:string?
format-time(xs:time?, xs:string) **as** xs:string?
function-available(xs:string) **as** xs:boolean
function-available(xs:string, xs:integer) **as** xs:boolean
generate-id() **as** xs:string
generate-id(node()?) **as** xs:string
key(xs:string, xs:anyAtomicType*) **as** node()*
key(xs:string, xs:anyAtomicType*, node()) **as** node()*
regex-group(xs:integer) **as** xs:string
system-property(xs:string) **as** xs:string
type-available(xs:string) **as** xs:boolean
unparsed-text(xs:string?) **as** xs:string?
unparsed-text(xs:string?, xs:string) **as** xs:string?
unparsed-text-available(xs:string?) **as** xs:boolean
unparsed-text-available(xs:string?, xs:string?) **as** xs:boolean
unparsed-entity-uri(xs:string) **as** xs:anyURI
unparsed-entity-public-id(xs:string) **as** xs:string

Argument Notation

numeric	Any of xs:integer, xs:decimal, xs:float or xs:double.
*	A sequence of the indicated type.
?	The indicated type or empty sequence.
~	The result type varies depending on the arguments.
xs:	http://www.w3.org/2001/XMLSchema

Geben Sie die Namen der Professoren zusammen mit den Namen der Studenten aus, die eine Vorlesung bei ihnen hören.

```
for $v in //Vorlesung ,
    $s in //Student
where contains($s/@hoert, $v/@VorlNr)
return
<ProfStud>
  <Prof>{data($v/../../Name)}</Prof>
  <Stud>{data($s/Name)}</Stud>
</ProfStud>
```

XQuery 1.0 & XPath 2.0

XQuery

BaseX:

```
for $r in //ProfessorIn return $r
```

MS-SQL:

```
SELECT doc.query('
  for $r in //ProfessorIn return $r
') AS Result
FROM uni
```

Contains | fn:id()

```
for $v in //Vorlesung return
  <Vorlesung>{$v/Titel}
    <Voraussetzungen>
    { for $e in //Vorlesung
      where contains($v/@Voraussetzungen, $e/@VorlNr)
      return <VorgaengerTitel>{$e/Titel}</VorgaengerTitel>}
    </Voraussetzungen>
  </Vorlesung>
```

```
for $v in //Vorlesung return
  <Vorlesung>{$v/Titel}
    <Voraussetzungen>
      {fn:id($v/@Voraussetzungen)/Titel}
    </Voraussetzungen>
  </Vorlesung>
```

Geben Sie Titel und Anzahl SWS aller Vorlesungen aus. Diese sind nach SWS sortiert (zuerst die Vorlesungen mit 4 SWS, dann die mit 3 SWS, dann die mit 2 SWS).

```
for $v in //Vorlesung
let $orderVar := $v/SWS
order by $orderVar descending
return
  <Vorlesung>{$v/Titel}{$v/SWS}</Vorlesung>
```

Geben Sie die Namen der Professoren aus, die eine Vorlesung halten, die vom Studenten „Carnap“ besucht wird.

```
<Professoren>
{
  for $v in //Vorlesung,
      $s in //Student[Name = "Carnap"]
  where contains($s/@hoert, $v/@VorlNr)
  return
    $v/../../Name
}
</Professoren>
```

Welche Studenten besuchen die Vorlesung „Grundzüge“?

```
for $v in //Vorlesung[Titel = "Grundzuege"]
return
  <Vorlesung>
    {$v/Titel}
    <Studenten>
    {
      for $s in //Student
      where contains($s/@hoert, $v/@VorlNr)
      return
        $s/Name
    }
    </Studenten>
  </Vorlesung>
```

Date/Time Operators

(xs:date) + (xs:dayTimeDuration) **as** xs:date
(xs:date) + (xs:yearMonthDuration) **as** xs:date
(xs:dateTime) + (xs:dayTimeDuration) **as** xs:dateTime
(xs:dateTime) + (xs:yearMonthDuration) **as** xs:dateTime
(xs:dayTimeDuration) + (xs:dayTimeDuration) **as** xs:dayTimeDuration
(xs:time) + (xs:dayTimeDuration) **as** xs:time
(xs:yearMonthDuration) + (xs:yearMonthDuratior **as** xs:yearMonthDuration
(xs:date) – (xs:date) **as** xs:dayTimeDuration
(xs:date) – (xs:dayTimeDuration) **as** xs:date
(xs:date) – (xs:yearMonthDuration) **as** xs:date
(xs:dateTime) – (xs:dateTime) **as** xs:dayTimeDuration
(xs:dateTime) – (xs:dayTimeDuration) **as** xs:dateTime
(xs:dateTime) – (xs:yearMonthDuration) **as** xs:dateTime
(xs:dayTimeDuration) – (xs:dayTimeDuration) **as** xs:dayTimeDuration
(xs:time) – (xs:dayTimeDuration) **as** xs:time
(xs:time) – (xs:time) **as** xs:dayTimeDuration
(xs:yearMonthDuration) – (xs:yearMonthDuration **as** xs:yearMonthDuration
(xs:dayTimeDuration) * (xs:double) **as** xs:dayTimeDuration
(xs:yearMonthDuration) * (xs:double) **as** xs:yearMonthDuration
(xs:dayTimeDuration) **div** (xs:dayTimeDuration) **a** xs:decimal
(xs:dayTimeDuration) **div** (xs:double) **as** xs:dayTimeDuration
(xs:yearMonthDuration) **div** (xs:double) **as** xs:yearMonthDuration
(xs:yearMonthDuration) **div** (xs:yearMonthDuration) **as** xs:decimal

The **eq, ne, lt, gt, le** and **ge** comparisons are suppoted for the types: **xs:date** and **xs:time.**

The **eq** and **ne** (only) comparisons are supported for the types: **xs:duration**, **xs:gDay**, **xs:gMonth**, **xs:gMonthDay**, **xs:gYear** and **xs:gYearMonth.**

The **lt, gt, le** and **ge** (only) comparisons are supported for the types: **xs:dayTimeDuration** and **xs:yearMonthDuration.**

Other Comparisons

The **eq** and **ne** (only) comparisons are supported for the types: **xs:base64Binary**, **xs:hexBinary**, **xs:NOTATION** and **xs:QName.**

Geben Sie den Namen aller Studenten zusammen mit einer Liste der von ihnen besuchten Vorlesungen aus.

```
for $s in //Student ,
    $v in //Vorlesung
where contains($s/@hoert, $v/@VorlNr)
return
  <Student>
    {$s/Name}
    <Vorlesung>{$v/Titel}</Vorlesung>
  </Student>
```

Text/String Functions

codepoint-equal(xs:string?, xs:string?) **as** xs:boolean?
codepoints-to-string(xs:integer*) **as** xs:string
compare(xs:string?, xs:string?) **as** xs:integer?
compare(xs:string?, xs:string?, xs:string) **as** xs:integer?
concat(xs:anyAtomicType?, xs:anyAtomicType?,) **as** xs:string
contains(xs:string?, xs:string?) **as** xs:boolean
contains(xs:string?, xs:string?, xs:string) **as** xs:boolean
current-date() **as** xs:date
current-dateTime() **as** xs:dateTime
current-time() **as** xs:time
default-collation() **as** xs:string
encode-for-uri(xs:string?) **as** xs:string
ends-with(xs:string?, xs:string?) **as** xs:boolean
ends-with(xs:string?, xs:string?, xs:string) **as** xs:boolean
escape-html-uri(xs:string?) **as** xs:string
lower-case(xs:string?) **as** xs:string
normalize-space() **as** xs:string
normalize-space(xs:string?) **as** xs:string
normalize-unicode(xs:string?) **as** xs:string
normalize-unicode(xs:string?, xs:string) **as** xs:string
starts-with(xs:string?, xs:string?) **as** xs:boolean
starts-with(xs:string?, xs:string?, xs:string) **as** xs:boolean
string() **as** xs:string
string(item()?) **as** xs:string
string-join(xs:string*, xs:string) **as** xs:string
string-length() **as** xs:integer
string-length(xs:string?) **as** xs:integer
string-to-codepoints(xs:string?) **as** xs:integer*
substring(xs:string?, xs:double) **as** xs:string
substring(xs:string?, xs:double, xs:double) **as** xs:string
substring-after(xs:string?, xs:string?) **as** xs:string
substring-after(xs:string?, xs:string?, xs:string) **as** xs:string
substring-before(xs:string?, xs:string?) **as** xs:string
substring-before(xs:string?, xs:string?, xs:string) **as** xs:string
translate(xs:string?, xs:string, xs:string) **as** xs:string
upper-case(xs:string?) **as** xs:string

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

REGEX Functions

matches(xs:string?, xs:string) **as** xs:boolean
matches(xs:string?, xs:string, xs:string) **as** xs:boolean
replace(xs:string?, xs:string, xs:string) **as** xs:string
replace(xs:string?, xs:string, xs:string, xs:string) **as** xs:string
tokenize(xs:string?, xs:string) **as** xs:string*
tokenize(xs:string?, xs:string, xs:string) **as** xs:string*

Arithmetic Operators

+ (numeric) **as** ~numeric
(numeric) + (numeric) **as** ~numeric
- (numeric) **as** ~numeric
(numeric) - (numeric) **as** ~numeric
(numeric) * (numeric) **as** ~numeric
(numeric) **div** (numeric) **as** ~numeric
(numeric) **idiv** (numeric) **as** xs:integer
(numeric) **mod** (numeric) **as** ~numeric

Arithmetic Functions

abs(numeric?) **as** ~numeric?
avg(xs:anyAtomicType*) **as** ~xs:anyAtomicType?
ceiling(numeric?) **as** ~numeric?
floor(numeric?) **as** ~numeric?
number() **as** xs:double
number(xs:anyAtomicType?) **as** xs:double
round(numeric?) **as** ~numeric?
round-half-to-even(numeric?) **as** ~numeric?
round-half-to-even(numeric?, xs:integer) **as** ~numeric?
sum(xs:anyAtomicType*) **as** ~xs:anyAtomicType
sum(xs:anyAtomicType*, xs:anyAtomicType?) **as** ~xs:anyAtomicType?

The **eq, ne, lt, gt, le** and **ge** comparisons are supported for the numeric types.

Sequence Operators

(item()*) **,** (item()*) **as** ~item()*
(node()*) **union** (node()*) **as** ~node()*
(node()*) **intersect** (node()*) **as** ~node()*
(node()*) **except** (node()*) **as** ~node()*
(xs:integer) **to** (xs:integer) **as** xs:integer*

Node Comparisons

(node()) **is** (node()) **as** xs:boolean
(node()) << (node()) **as** xs:boolean
(node()) >> (node()) **as** xs:boolean

Sequence and Node Functions

collection() **as** node()*
collection(xs:string?) **as** node()*
count(item()*) **as** xs:integer
data(item()*) **as** ~xs:anyAtomicType*
deep-equal(item()*, item()*) **as** xs:boolean
deep-equal(item()*, item()*, string) **as** xs:boolean
distinct-values(xs:anyAtomicType*) **as** ~xs:anyAtomicType*
distinct-values(xs:anyAtomicType*, xs:string) **as** ~xs:anyAtomicType*
doc(xs:string?) **as** document-node()?
empty(item()*) **as** xs:boolean
exactly-one(item()*) **as** ~item()
exists(item()*) **as** xs:boolean
index-of(xs:anyAtomicType*, xs:anyAtomicType) **as** xs:integer*
index-of(xs:anyAtomicType*, xs:anyAtomicType, xs:string) **as** xs:integer*
insert-before(item()*, xs:integer, item()*) **as** ~item()*
last() **as** xs:integer
nilled(node()?) **as** xs:boolean?
node-name(node()?) **as** xs:QName?
one-or-more(item()*) **as** ~item()+
position() **as** xs:integer
remove(item()*, xs:integer) **as** ~item()*
reverse(item()*) **as** ~item()*
root() **as** node()
root(node()?) **as** node()?
subsequence(item()*, xs:double) **as** ~item()*
subsequence(item()*, xs:double, xs:double) **as** ~item()*
unordered(item()*) **as** ~item()*
zero-or-one(item()*) **as** ~item()?

Miscellaneous Functions

error() **as** none
error(xs:QName) **as** none
error(xs:QName?, xs:string) **as** none
error(xs:QName?, xs:string, item()*) **as** none
lang(xs:string?) **as** xs:boolean
lang(xs:string?, node()) **as** xs:boolean
max(xs:anyAtomicType*) **as** ~xs:anyAtomicType?
max(xs:anyAtomicType*, string) **as** ~xs:anyAtomicType?
min(xs:anyAtomicType*) **as** ~xs:anyAtomicType?
min(xs:anyAtomicType*, string) **as** ~xs:anyAtomicType?
trace(item()*, xs:string) **as** ~item()*

Boolean Functions

boolean(item()*) **as** xs:boolean
false() **as** xs:boolean
not(item()*) **as** xs:boolean
true() **as** xs:boolean

The **eq, ne, lt, gt, le** and **ge** comparisons are supported for the **xs:boolean** type.

URI, ID and XML Name Functions

base-uri() **as** xs:anyURI?
base-uri(node()?) **as** xs:anyURI?
document-uri(node()?) **as** xs:anyURI?
doc-available(xs:string?) **as** xs:boolean
in-scope-prefixes(element()) **as** xs:string*
id(xs:string*) **as** element()*
id(xs:string*, node()) **as** element()*
idref(xs:string*) **as** node()*
idref(xs:string*, node()) **as** node()*
iri-to-uri(xs:string?) **as** xs:string
local-name() **as** xs:string
local-name(node()?) **as** xs:string
local-name-from-QName(xs:QName?) **as** xs:NCName?
name() **as** xs:string
name(node()?) **as** xs:string
namespace-uri() **as** xs:anyURI
namespace-uri(node()?) **as** xs:anyURI
namespace-uri-for-prefix(xs:string?, element()) **as** xs:anyURI?
namespace-uri-from-QName(xs:QName?) **as** xs:anyURI?
prefix-from-QName(xs:QName?) **as** xs:NCName?
QName(xs:string?, xs:string) **as** xs:QName
resolve-QName(xs:string?, element()) **as** xs:QName?
resolve-uri(xs:string?) **as** xs:anyURI?
resolve-uri(xs:string?, xs:string) **as** xs:anyURI?
static-base-uri() **as** xs:anyURI?

Built-In Schema Types

These types are available in all implementations.

xs:anyAtomicType	xs:gMonth
xs:anySimpleType	xs:anyURI
xs:anyType	xs:gMonthDay
xs:base64Binary	xs:gYear
xs:boolean	xs:gYearMonth
xs:date	xs:hexBinary
xs:dateTime	xs:integer
xs:dayTimeDuration	xs:QName
xs:decimal	xs:string
xs:double	xs:time
xs:duration	xs:untyped
xs:float	xs:untypedAtomic
xs:gDay	xs:yearMonthDuration