

XSLT 4.0



Extra Features

- Enhancements to the type system to allow more expressive constraints, especially for maps and atomic values.
- Additional functionality for processing arrays.
- Exploitation of the power afforded by first-class function items



XSLT 4.0

- `xsl:item-type`
- For-each and apply-templates separator
- `mode/@as`
- Extension instructions
- `map/@duplicates` cf `map:build()`
- `Xsl:for-each-group/@split-when`
- Named record types
- `Xsl:record`
- `@select` on `xsl:map|result-document|when|otherwise|(non-)matching-substring`
- `Xsl:select`
- Self-referential Global variables of `function()` type
- Parameters passed automatically to `apply-imports` and `next-match`
- Optional parameters to `xsl:function`
- `Xsl:switch`
- `Xsl:note stylesheet/@main-module`
- `stylesheet/@fixed-namespaces`
- Type-based patterns
- `Xsl:array`
- Enclosing mode
- Shallow-copy-all for map/array trees
- Ordered maps (XPath)



Syntax additions

- Many of the XSLT syntax additions can be source-converted to XSLT3
- A browser-based workbench can show these and execute

The screenshot shows the jwL QT4 workbench interface in a web browser. The browser address bar shows the URL `localhost/mine/DA2025/jwLQT4.xhtml`. The interface is divided into three main sections:

- Stylesheet:** Contains XSLT source code. The code includes a namespace declaration for `http://www.w3.org/1999/XSL/Transform` and a template named `reverse` that applies to the `author` element. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" fixed-
namespaces="#default"
  exclude-result-prefixes="#all" version="4.0">

  <xsl:template match="/">
    <xsl:apply-templates select="." mode="reverse"/>
  </xsl:template>
  <xsl:mode name="reverse" on-no-match="shallow-copy">
    <xsl:template match="author">
      <writer>
        <xsl:apply-templates/>
      </writer>
    </xsl:template>
  </xsl:mode>
</xsl:stylesheet>
```
- Input:** Contains the input XML document. The XML is as follows:

```
<ContactTitle>Owner</ContactTitle>
<Phone>(26) 642-7012</Phone>
<Fax>(26) 642-7012</Fax>
<FullAddress>
  <Address>ul. Filtrowa 68</Address>
  <City>Warszawa</City>
  <PostalCode>01-012</PostalCode>
  <Country>Poland</Country>
</FullAddress>
</Customers>
</Root>
```
- Result:** Contains the output XML document. The output is as follows:

```
<Root>
  <Customers CustomerID="ALFKI">
    <CompanyName>Alfreds Futterkiste</CompanyName>
    <ContactName>Maria Anders</ContactName>
    <ContactTitle>Sales Representative</ContactTitle>
    <Phone>030-0074321</Phone>
    <Fax>030-0076545</Fax>
    <FullAddress>
      <Address>Obere Str. 57</Address>
      <City>Berlin</City>
      <PostalCode>12209</PostalCode>
      <Country>Germany</Country>
    </FullAddress>
  </Customers>
  <Customers CustomerID="ANATR">
    <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
    <ContactName>Ana Trujillo</ContactName>
    <ContactTitle>Owner</ContactTitle>
    <Phone>(5) 555-4729</Phone>
    <Fax>(5) 555-3745</Fax>
    <FullAddress>
      <Address>Avda. de la Constitución 2222</Address>
      <City>México D.F.</City>
      <PostalCode>05021</PostalCode>
      <Country>Mexico</Country>
    </FullAddress>
  </Customers>
  <Customers CustomerID="ANTON">
    <CompanyName>Antonio Moreno Taqueria</CompanyName>
    <ContactName>Antonio Moreno</ContactName>
    <ContactTitle>Owner</ContactTitle>
    <Phone>(5) 555-3932</Phone>
    <FullAddress>
      <Address>Mataderos 2312</Address>
      <City>México D.F.</City>
      <PostalCode>05023</PostalCode>
      <Country>Mexico</Country>
    </FullAddress>
  </Customers>
</Root>
```

At the bottom of the interface, there is a **GO!** button and a section for **Options** and **Result**. The **Options** section includes a checkbox for **Show Advanced options/actions**. The **Result** section includes a checkbox for **Indent result (may add whitespace)**, which is currently checked.



XSLT 4.0

Syntax additions – increasing XSLT code density

```
<xsl:choose>
  <xsl:when test="$n lt $low">
    <xsl:sequence select="../below"/>
  </xsl:when>
  <xsl:when test="$n gt $high">
    <xsl:sequence select="../above"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="../within"/>
  </xsl:otherwise>
</xsl:choose>
```

XSLT 3.0 code

Terser XSLT 4.0 equivalent

```
<xsl:choose>
  <xsl:when test="$n lt $low" select="../below"/>
  <xsl:when test="$n gt $high" select="../above"/>
  <xsl:otherwise select="../within"/>
</xsl:choose>
```



XSLT 4.0

Selection – @select vs xsl:sequence[@select]

```
<xsl:choose>
  <xsl:when test="$n lt $low">
    <xsl:sequence select="../below"/>
  </xsl:when>
  <xsl:when test="$n gt $high">
    <xsl:sequence select="../above"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="../within"/>
  </xsl:otherwise>
</xsl:choose>
```

Similarly for:
xsl:map |
xsl:result-document |
xsl:matching-substring |
xsl:non-matching-substring

```
<xsl:choose>
  <xsl:when test="$n lt $low" select="../below"/>
  <xsl:when test="$n gt $high" select="../above"/>
  <xsl:otherwise select="../within"/>
</xsl:choose>
```

Selection – preserving formatted XPath expressions

```
<xsl:function name="f:book-to-json" as="map(*)">
  <xsl:param name="book" as="element(book)">
    <xsl:select>
      {
        "title"   : string($book/title),
        "isbn"    : string($book/isbn),
        "price"   : number($book/@price),
        "author"  : array { $book/author ! string() },
        "date"    : current-date()
      }
    </xsl:select>
  </xsl:function>
```



XSLT 4.0

If @then @else

```
<xsl:choose>
  <xsl:when test="$n ge $low and $n le $high">
    <xsl:sequence select="../within"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="../without"/>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:if test="$n ge $low and $n le $high"
  then="../within" else="../without"/>
```




Enclosing Mode

```
<xsl:mode name="removePara" on-no-match="shallow-copy"/>
```

```
<xsl:template mode="removePara" as="element(*)" match="/">  
  <xsl:apply-templates mode="#current"/>  
</xsl:template>
```

Lots of other stuff which makes finding all relevant matches difficult

```
<xsl:template mode="removePara" as="element(*)" match="p"/>
```

```
<xsl:mode name="removePara" on-no-match="shallow-copy" as="element(*)">  
  <xsl:template match="/">  
    <xsl:apply-templates/>  
  </xsl:template>  
  <xsl:template match="p"/>  
</xsl:mode>
```



Optional function parameters

```
<xsl:function name="f:inRange">
  <xsl:param name="n"/>
  <xsl:sequence select="f:inRange($n, 0)"/>
</xsl:function>
<xsl:function name="f:inRange">
  <xsl:param name="n"/>
  <xsl:param name="low"/>
  <xsl:variable name="high">
    <xsl:call-template name="findLimit"/>
  </xsl:variable>
  <xsl:sequence select="f:inRange($n, $low, $high)"/>
</xsl:function>
<xsl:function name="f:inRange">
  <xsl:param name="n"/>
  <xsl:param name="low"/>
  <xsl:param name="high"/>
  <!-- Stuff -->
</xsl:function>
```

```
<xsl:function name="f:inRange">
  <xsl:param name="n" as="element()"/>
  <xsl:param name="low" required="no" select="0"/>
  <xsl:param name="high" required="no">
    <xsl:call-template name="findLimit"/>
  </xsl:param>
  <!-- Stuff -->
</xsl:function>
```

Named Item Types

```
<xsl:param name="upscale" as="map(xs:string,function(xs:double) as xs:double)"/>
<xsl:param name="downscale" as="map(xs:string,function(xs:double) as xs:double)"/>
<xsl:variable name="converters" as="map(xs:string,function(xs:double) as xs:double)"
  select="map{
    'double': function ($x) {$x + $x},
    'quadruple': function ($x) {4 * $x}
  }"/>
```

```
<xsl:item-type name="f:converters"
  as="map(xs:string,function(xs:double) as xs:double)"/>

<xsl:param name="upscale" as="f:converters"/>
<xsl:param name="downscale" as="f:converters"/>
<xsl:variable name="converters" as="f:converters" select="
  map {
    'double': function ($x) {$x + $x},
    'quadruple': function ($x) {4 * $x}
  }"/>
```

Record Types, Constructors

```
<xsl:record-type name="f:complex" constructor="true">
  <xsl:field name="r" as="xs:double"/>
  <xsl:field name="i" as="xs:double" required="no" default="0"/>
</xsl:record-type>

<xsl:variable name="i" as="f:complex" select="f:complex(0, 1)"/>
```

```
<xsl:record xsl:as="f:complex" r="15" i="12" xsl:use-when="$use-records"/>
```



Extension Instructions to call templates

```
<xsl:template name="f:report">
  <xsl:param name="code" as="xs:integer" required="yes"/>
  <xsl:param name="message" as="xs:string?"/>
  <report code="{ $code }">{ ( $message, 'Emergency' ) [1] }</report>
</xsl:template>
```

```
<xsl:call-template name="f:report">
  <xsl:with-param name="code" select="123"/>
  <xsl:with-param name="message" select="'All normal'"/>
</xsl:call-template>
```

...

... extension-element-prefixes="f"...

```
<xsl:template name="f:report"> ... </xsl:template/>
```

```
<f:report code="123" message="All normal"/>
```

```
<f:report code="999"/>
```

Fixed namespaces

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:array="http://www.w3.org/2005/xpath-functions/array"
  xmlns:err="http://www.w3.org/2005/xqt-errors"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:map="http://www.w3.org/2005/xpath-functions/map"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="3.0">
  <!-- Stuff -->
</xsl:stylesheet>
```

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  fixed-namespaces="#default"
  version="4.0"
  <!-- Stuff -->
</xsl:stylesheet>
```



XSLT 4.0

Separators

```
<xsl:apply-templates select="AUTHOR">
  <xsl:sort select="LAST-NAME"/>
  <xsl:sort select="FIRST-NAME"/>
</xsl:apply-templates>

<xsl:template match="AUTHOR">
  <!-- Stuff -->
  <xsl:if test="position() lt last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:for-each>
```

```
<xsl:for-each select="AUTHOR">
  <xsl:sort select="LAST-NAME"/>
  <xsl:sort select="FIRST-NAME"/>
  <xsl:apply-templates select="."/>
  <xsl:if test="position() lt last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:for-each>
```

```
<xsl:apply-templates select="AUTHOR" separator=", ">
  <xsl:sort select="LAST-NAME"/>
  <xsl:sort select="FIRST-NAME"/>
</xsl:apply-templates>
```

Similarly for:
xsl:for-each



Switch

```
<xsl:variable name="n" select="a" as="item()"/>
<xsl:choose>
  <xsl:when test="$n = (1, 2, 3, 4)">
    <xsl:sequence select="../small"/>
  </xsl:when>
  <xsl:when test="$n = (5 to 20)">
    <xsl:sequence select="../medium"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="../large"/>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:switch select="$n">
  <xsl:when test="1, 2, 3, 4" select="../small"/>
  <xsl:when test="5 to 20" select="../medium"/>
  <xsl:otherwise select="../large"/>
</xsl:switch>
```




Arrays FIX!!!!!!

```
<xsl:variable name="n" select="a" as="item()"/>
<xsl:choose>
  <xsl:when test="$n = (1, 2, 3, 4)">
    <xsl:sequence select="../small"/>
  </xsl:when>
  <xsl:when test="$n = (5 to 20)">
    <xsl:sequence select="../medium"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="../large"/>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:switch select="$n">
  <xsl:when test="1, 2, 3, 4" select="../small"/>
  <xsl:when test="5 to 20" select="../medium"/>
  <xsl:otherwise select="../large"/>
</xsl:switch>
```



Development/documentation aid

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  main-module="toplevel-uri.xsl" ...>

  <xsl:note role="description">
    <p>This is a note <b>purely</b> for purposes outside XSLT. <br/>
    Think of it as a comment that can contain structured stuff.</p>
  </xsl:note>
  <xsl:function name="t:temp">
    <xsl:note class="simple">It can appear anywhere as a child element
    and can have any attributes ...
  </xsl:note>
  <xsl:param name="a">
    <xsl:note continuation-of="previous::xsl:note[1]">
      ... and is totally ignored by XSLT for any
      syntactic or semantic purposes.</xsl:note>
    </xsl:param>
    ...
  </xsl:function>
</xsl:stylesheet>
```