

# 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



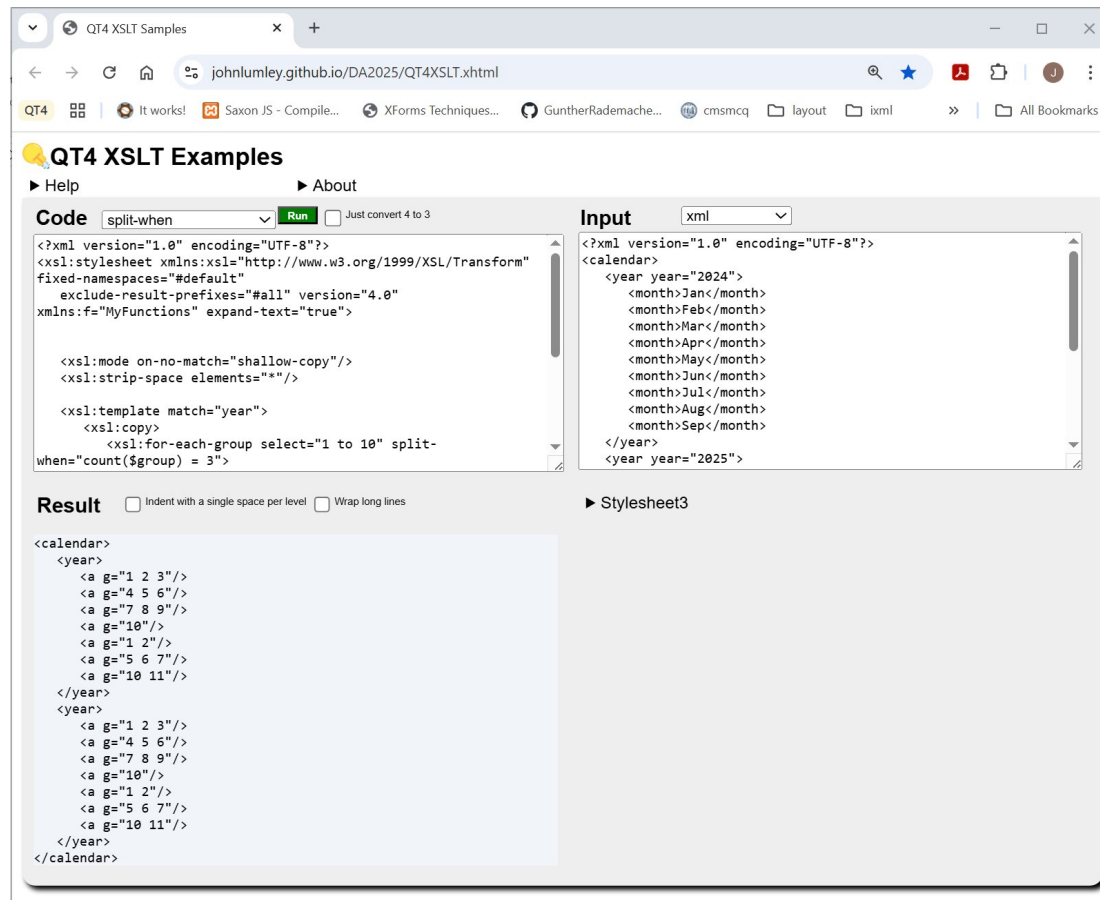
## XSLT 4.0

- `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 does this and executes





**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**

Examples will be simple,  
possibly nonsensical, but  
illustrative

```
<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 i

## @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>
```



XSLT 4.0

# Selection ii

## preserving formatted XPath expressions

```
<xsl:function name="f:book-to-json" as="map(*)">
  <xsl:param name="book" as="element(book)">
    <xsl:sequence select="
      { 'title' : string($book/title),
        'author' : array { $book/author ! string() },
        'date' : let $d := $book/publication return
                    if(current-date() lt xs:date($d))
                    then $d else 'Don&apos;t release'
      }"/>
</xsl:function>
```

Attribute value  
normalization



```
<xsl:function name="f:book-to-json" as="map(*)">
  <xsl:param name="book" as="element(book)">
    <xsl:sequence select="{ 'title' : string($book/title),
      'author' : array { $book/author ! string() }, 'date' : let
        $d := $book/publication return if(current-date() lt
        xs:date($d)) then $d else 'Don&apos;t release' }"/>
</xsl:function>
```

# Selection iii

## 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),
        'author'  : array { $book/author ! string() },
        'date'    : let $d := $book/publication return
                     if(current-date() lt xs:date($d))
                     then $d else "Don't release"
      }
    </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)"/>
```

`$i`  `{'r': 0, 'i': 10}`

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

 `{'r': 15, 'i': 12}`

This is an exception – standard  
attributes **must** be in XSLT namespace



# 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"/>
```

Example  
**Callable**  
extension call

# 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>
```

# Grouping – split-when

```
<xsl:for-each-group select="1 to 4, 5, 7 to 9, 10 to 12, 14, 15, 20 to 22"  
  split-when="$next ne ($group[last()] + 1)">  
  <a g="{current-group()}" />  
</xsl:for-each-group>
```

Example  
Grouping  
split-when



## XSLT 4.0

# 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>
```