



XSLT 4.0



Main Features

- Syntax additions to make code 'terser'
- Enhancements to the type system to allow more expressive constraints, especially for maps and atomic values.
- Type-based pattern matching
- Additional functionality for processing arrays
- Streaming split from the main specification



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>
```



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

These slide examples will be simple,
possibly nonsensical or easier in
direct XPath, 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'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't release' }"/>
  </xsl:function>
```



XSLT 4.0

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

xsl:if @then @else

```
<xsl:choose>
  <xsl:when test="$n ge 0">
    <xsl:sequence select="1"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="-1"/>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:choose>
  <xsl:when test="$n ge $low and $n le $high">
    <xsl:apply-templates/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:sequence select="
      error(), 'Outside limits')"/>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:if test="$n ge 0"
  then="1" else "-1"/>
```

```
<xsl:if test="$n ge $low and $n le $high"
  else="error(), 'Outside limits')">
  <xsl:apply-templates/>
</xsl:if>
```



XSLT 4.0

```
<xsl:variable name="n" select="a"/>
<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>
```

Switch

```
<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>
```



XSLT workbench – i

The screenshot shows the QT4 XSLT Samples application interface. On the left, there's a sidebar with "QT4 XSLT Examples" and links for "Help" and "About". Below the sidebar, there's a note about using `xsl:array` with `@for-each` to generate lists of duplicate words for each para, e.g.:

```
<output>
<duplicates para="1">and by it The to of
<duplicates para="2">are and the by transforming JSON</duplicates>
</output>
```

On the right, there's a note about finding parallel functions to new ones in F&O: `f:duplicate-values()` and `ar:members()`.

XSLT 4 CODE

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" fixed-namespaces="#default"
  xmlns:f="MyFunctions" xmlns:ar="MyArray" exclude-result-prefixes="#all" expand-text="true"
  version="4.0">

  <xsl:template match="/">
    <xsl:variable name="duplicates" select="array</xsl:variable>
    <xsl:for-each select="array"></xsl:for-each>
      <xsl:variable name="r">array</xsl:variable>
      <xsl:for-each select="r:members($duplicates)">
        <xsl:if test="count(.?value) > 0">
          <xsl:for-each select="array"></xsl:for-each>
            <xsl:variable name="r2">array</xsl:variable>
            <xsl:for-each select="r2:array"></xsl:for-each>
              <xsl:variable name="r3">array</xsl:variable>
              <xsl:for-each select="r3:array"></xsl:for-each>
                <xsl:variable name="r4">array</xsl:variable>
                <xsl:for-each select="r4:array"></xsl:for-each>
                  <xsl:variable name="r5">array</xsl:variable>
                  <xsl:for-each select="r5:array"></xsl:for-each>
                    <xsl:variable name="r6">array</xsl:variable>
                    <xsl:for-each select="r6:array"></xsl:for-each>
                      <xsl:variable name="r7">array</xsl:variable>
                      <xsl:for-each select="r7:array"></xsl:for-each>
                        <xsl:variable name="r8">array</xsl:variable>
                        <xsl:for-each select="r8:array"></xsl:for-each>
                          <xsl:variable name="r9">array</xsl:variable>
                          <xsl:for-each select="r9:array"></xsl:for-each>
                            <xsl:variable name="r10">array</xsl:variable>
                            <xsl:for-each select="r10:array"></xsl:for-each>
                              <xsl:variable name="r11">array</xsl:variable>
                              <xsl:for-each select="r11:array"></xsl:for-each>
                                <xsl:variable name="r12">array</xsl:variable>
                                <xsl:for-each select="r12:array"></xsl:for-each>
                                  <xsl:variable name="r13">array</xsl:variable>
                                  <xsl:for-each select="r13:array"></xsl:for-each>
                                    <xsl:variable name="r14">array</xsl:variable>
                                    <xsl:for-each select="r14:array"></xsl:for-each>
                                      <xsl:variable name="r15">array</xsl:variable>
                                      <xsl:for-each select="r15:array"></xsl:for-each>
                                        <xsl:variable name="r16">array</xsl:variable>
                                        <xsl:for-each select="r16:array"></xsl:for-each>
                                          <xsl:variable name="r17">array</xsl:variable>
                                          <xsl:for-each select="r17:array"></xsl:for-each>
                                            <xsl:variable name="r18">array</xsl:variable>
                                            <xsl:for-each select="r18:array"></xsl:for-each>
                                              <xsl:variable name="r19">array</xsl:variable>
                                              <xsl:for-each select="r19:array"></xsl:for-each>
                                                <xsl:variable name="r20">array</xsl:variable>
                                                <xsl:for-each select="r20:array"></xsl:for-each>
                                                  <xsl:variable name="r21">array</xsl:variable>
                                                  <xsl:for-each select="r21:array"></xsl:for-each>
                                                    <xsl:variable name="r22">array</xsl:variable>
                                                    <xsl:for-each select="r22:array"></xsl:for-each>
                                                      <xsl:variable name="r23">array</xsl:variable>
                                                      <xsl:for-each select="r23:array"></xsl:for-each>
                                                        <xsl:variable name="r24">array</xsl:variable>
                                                        <xsl:for-each select="r24:array"></xsl:for-each>
                                                          <xsl:variable name="r25">array</xsl:variable>
                                                          <xsl:for-each select="r25:array"></xsl:for-each>
                                                            <xsl:variable name="r26">array</xsl:variable>
                                                            <xsl:for-each select="r26:array"></xsl:for-each>
                                                              <xsl:variable name="r27">array</xsl:variable>
                                                              <xsl:for-each select="r27:array"></xsl:for-each>
                                                                <xsl:variable name="r28">array</xsl:variable>
                                                                <xsl:for-each select="r28:array"></xsl:for-each>
                                                                  <xsl:variable name="r29">array</xsl:variable>
                                                                  <xsl:for-each select="r29:array"></xsl:for-each>
                                                                    <xsl:variable name="r30">array</xsl:variable>
                                                                    <xsl:for-each select="r30:array"></xsl:for-each>
                                                                      <xsl:variable name="r31">array</xsl:variable>
                                                                      <xsl:for-each select="r31:array"></xsl:for-each>
                                                                        <xsl:variable name="r32">array</xsl:variable>
                                                                        <xsl:for-each select="r32:array"></xsl:for-each>
                                                                          <xsl:variable name="r33">array</xsl:variable>
                                                                          <xsl:for-each select="r33:array"></xsl:for-each>
                                                                            <xsl:variable name="r34">array</xsl:variable>
                                                                            <xsl:for-each select="r34:array"></xsl:for-each>
                                                                              <xsl:variable name="r35">array</xsl:variable>
                                                                              <xsl:for-each select="r35:array"></xsl:for-each>
                                                                                <xsl:variable name="r36">array</xsl:variable>
                                                                                <xsl:for-each select="r36:array"></xsl:for-each>
                                                                                  <xsl:variable name="r37">array</xsl:variable>
                                                                                  <xsl:for-each select="r37:array"></xsl:for-each>
                                                                                    <xsl:variable name="r38">array</xsl:variable>
                                                                                    <xsl:for-each select="r38:array"></xsl:for-each>
                                                                                      <xsl:variable name="r39">array</xsl:variable>
                                                                                      <xsl:for-each select="r39:array"></xsl:for-each>
                        </xsl:for-each>
                      </xsl:for-each>
                    </xsl:for-each>
                  </xsl:for-each>
                </xsl:for-each>
              </xsl:for-each>
            </xsl:for-each>
          </xsl:for-each>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:template>

```

RESULT

```
<output>
<duplicates para="1">and by it The to of
<duplicates para="2">are and the by
transforming JSON</duplicates>
<duplicates para="3">the of that
in</duplicates>
<duplicates para="4">JSON a to the that XML is
be</duplicates>
<duplicates para="5"> the that is to
be</duplicates>
<duplicates para="6">is for</duplicates>
<duplicates para="7">JSON languages is
manipulating</duplicates>
<duplicates para="8">for JSON is
to</duplicates>
<duplicates
para="11">capabilities</duplicates>
<duplicates para="12">and json-to-xml() XML
etc</duplicates>
</output>
```

XSLT 3 CODE

```
<xsl:stylesheet xmlns:xs="http://www.w3.org/2005/xpath-functions/array"
  xmlns:err="http://www.w3.org/2005/xpath-errors"
  xmlns:f="MyFunctions"
  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:xi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  exclude-result-prefixes="#all" expand-text="true"
  version="4.0">

  <!--DO NOT EDIT -
generated from
  http://localhost/mine/DA2025/QT4XSLT.3.sef.json
at 2025-10-26T11:24:18.96Z by
  2.6--><xsl:ext:
  --><xsl:transform:FourToThree-mode--><xsl:import
  href="../DA2025/QT4LibFO.3.xsl"/>
<xsl:template match="/">
```

SaxonJS only compiles/runs
XSLT 3 and XPath 3.1

Solution:

Convert XSLT 4 to ~equivalent XSLT 3 for a number of new constructs, and run via `fn:transform()`

Only XSLT syntax is converted

- XPath expressions are **not** converted – XPath 4.0 syntax additions and new/modified functions not supported
- Some ~equivalent functions are provided in `f:` `ar:` and `ma:` namespaces



XSLT 4.0

XSLT workbench – ii

Syntax limitations

map{'a':1,'b':2}

map keyword required
in 3, optional in 4

{'a':1,'b':2}

function(\$x) {\$x}

inline function keyword
must be **function** in 3,
can be **fn** in 4

fn(\$x) {\$x}

if(a) then b
else ()

if/then/else must be
complete in 3,
can use **{exp}** in 4

if(a) {b}



XSLT 4.0

XSLT workbench – iii

~Equivalent functions

```
fn:for-each($input as item()*,  
            $action as fn(item(), xs:integer) as item()*)  
    ) as item()*
```

Sequence position
not supported in
XPath 3.1

```
<!--QT4FOLib - imported into the XSLT3 stylesheet-->  
<xsl:function name="f:for-each" as="item()*"/>  
  <xsl:param name="input" as="item()*"/>  
  <xsl:param name="fn" as="function(*)"/>  
  <xsl:sequence select="for-each-pair(  
    $input,  
    1 to count($input),  
    $fn)"/>  
</xsl:function>
```

```
<xsl:sequence select="f:for-each(  
    ('a', 'b', 'c'),  
    function($x, $pos) {$x || $pos}  
)"/>
```

→ 'a1', 'b2', 'c3'

Hints will be provided when such functions could be helpful;
namespace prefixes are already bound in the stylesheet

You may find the
~equivalent functions
[f:duplicate-values\(\)](#) and
[ar:members\(\)](#) helpful.



XSLT 4.0

Enclosing Mode

```
<xsl:mode name="M1" on-no-match="shallow-copy"/>  
  
<xsl:template mode="M1" as="element()*" match="p"/>
```

Lots of other stuff which makes finding all relevant matches difficult

```
<xsl:template mode="M1" as="element()*" match="/">  
  <xsl:apply-templates mode="#current"/>  
</xsl:template>
```

**Example
Mode**

enclosing modes

mode=M1
for matching
& *default mode*
for application

```
<xsl:mode name="M1" as="element()*"  
          on-no-match="shallow-copy">  
  <xsl:template match="p"/>  
  ...  
  <xsl:template match="/">  
    <xsl:apply-templates/>  
  </xsl:template>  
</xsl:mode>
```

```
<xsl:template match="*" mode="M1"/>
```

templates in this mode **must not**
appear elsewhere in the package

```
<xsl:template match="*" mode="#all"/>
```

only templates **inside the mode** are
applicable – even **#all** doesn't apply



Optional function parameters - i

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



Optional function parameters - ii

```
<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="findNumberLimit"/>
  </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 - i

```
<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>
```

Constructor function

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

\$oneReal, \$i → {'r': 0, 'i': 1}, {'r': 1, 'i': 0}



XSLT 4.0

Record Types - ii

```
<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>
```

Constructor instruction

```
<xsl:record r="$end?x - $start?x" i="$end?y - $start?y"/>
```

xs:NCName of key



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

Expression evaluated
in current context

```
<xsl:record xsl:as="f:complex" r="$end?x - $start?x" i="$end?y - $start?y"
             xsl:use-when="$use-records"/>
```

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



Type-based pattern matching

```
<xsl:template match=".  
  [. instance of array(xs:date)] |  
  [. instance of map(xs:string, *)]">
```

```
<xsl:template match="type(array(xs:date)) |  
  type(map(xs:string, *))">
```

```
<xsl:template match="array(xs:date) |  
  map(xs:string, *)">
```

```
record(first, last, *)[?location = 'UK']
```

matches a map with keys including **"first"** and **"last"**, and an entry **{"location": "UK"}**

```
array(record(first, last, *)[array:size(.) gt 0])
```

matches a non-empty array where every entry is a map with keys including **"first"** and **"last"**

```
jnode(books, array(record(Author, Title, *)))
```

matches any **JNode** of a map entry whose key is **"books"** and whose content is an array of records, each record having entries keyed **"Author"** and **"Title"** (with other entries also allowed)



XSLT 4.0

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

```
<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>
```

→ Michael Kay, John Lumley

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

Similarly for:
xsl:for-each



Grouping – split-when

Next item Group so far

```
<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>
```

→

```
<b g="1 2 3 4 5"/>
<b g="7 8 9 10 11 12"/>
<b g="14 15"/>
<b g="20 21 22"/>
```

Example
Grouping
split-when: line-wrap



xsl:array @for-each

```
<xsl:variable name="powers" select="  
    let $parts := (1 to 4)!  
        map{'value':(., . * ., . * . * .)}  
    return  
        array:for-each(array{$parts},  
            function($a){$a?value})"/>
```

[(1,1,1),
(2,4,8),
(3,9,27),
(4,16,64)]

```
<xsl:variable name="powers" as="array(*)">  
    <xsl:array for-each="1 to 4" select=". * ." />  
</xsl:variable>  
  
<powers>{array:for-each($powers, string-join(?, '-'))}</powers>
```

→ <powers>1-1-1 2-4-8 3-9-27 4-16-64</powers>

Caution – proposed but not yet fully discussed/agreed

Example
Array

array/@for-each
duplicate words