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

XSLT 4.0

Syntax additions – increasing XSLT code density

XSLT 3.0 code

```
<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>
```

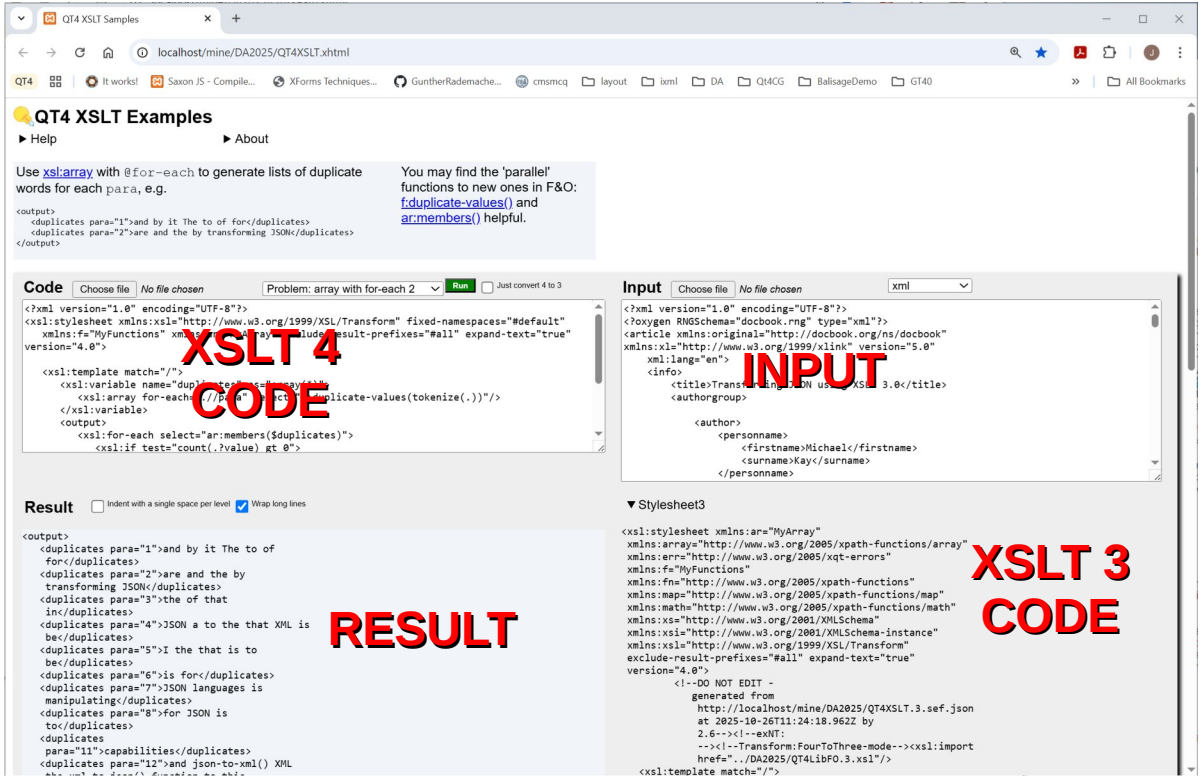
Terser XSLT 4.0 equivalent

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

These slide examples will be simple, possibly nonsensical or easier in direct XPath, but illustrative

XSLT 4.0

XSLT workbench – i



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.

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

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>
```

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

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

Switch

```
<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>
```

```
<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 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>
```

XSLT 4.0

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}
```

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”/>
```

Expression evaluated in current context

xs:NCName of key

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

```
<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

XSLT 4.0

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>
```

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`

XSLT 4.0

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

XSLT 4.0

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