



make it clever



**Query / Transformations 4.0**  
**Avancée des standards**

- **Christophe Marchand**

- **Développeur !**

- 24 ans de Java
- 30 ans d'objet
- 21 ans de XML
- Contributeur à XSpec



- **Formateur chez OXiane**

- Autour de Java, de XML, et de la qualité logicielle



- <https://github.com/cmarchand>



- <https://twitter.com/cmarchand7276>



- **Community Group du W3C**

- Michael Kay - Saxonica
- Norm Walsh - Saxonica
- Christian Grün - BaseX
- Dimitre Novatchev

- **Initiative de Michael Kay**



- XML Prague 2020 - Proposal for Xslt 4.0
- <https://www.xmlprague.cz/day2-2020/#xslt4>

# XPath Language 4.0

- **record**

- Permet de définir la structure d'une map : clés et types des valeurs



```
$v as record(*)  
  
$w as record(  
  r as xs:int, i as xs:int, j? as xs:int)
```

```
let v := {'a': 12, 'x': 'description'}  
  
let w := {'r': 21, 'i': 37}   
let w := {'r': 31, 'l': 53} 
```

- **enum**

- Une simple liste de **xs:string**

```
$color as enum('red','green','blue')
```

```
let color := 'blue'   
let color := 'pink' 
```

- Fonctions avec paramètres optionnels et valeur par défaut

```
fn:string-join(
  $values as xs:anyAtomicType*,
  $separator as xs:string? := ''
) as xs:string
```

\$separator peut être omis dans l'appel  
il vaudra alors ""

```
string-join( ('a','b','c') )      ⇒ 'abc'
string-join( ('a','b','c'), ' ') ⇒ 'a b c'
```

- On peut passer les paramètres par position ou par nom

```
string-join( ('a','b','c'), ' ')      ⇒ 'a b c'
string-join(separator := ' ', values := ('a','b','c')) ⇒ 'a b c'
```

- On peut mixer les 2 notations
  - Mais les positionnels doivent toujours être à la bonne position

- **Equivalent des Value-Template de Xslt**
  - Généralisé à XPath

```
let $greeting := "Hello",  
    $planet   := "Mars"  
  
return `{ $greeting }, { $planet }!`
```

# XPath & XQuery Functions & Operators



- Fonctions sur les String

```
fn:char(  
  $value as union(xs:string, xs:positiveInteger)  
) as xs:string
```

- Permet de générer un caractère sur la base de son nom ou de sa valeur unicode

char('\t')	-> caractère tabulation
char(0x20)	-> caractère espace
char('aacute')	-> 'á'

```
fn:characters(  
  $value as xs:string?  
) as xs:string*
```

- Découpe une chaîne de caractères en une séquence de caractères

```
characters('Hello') -> ('H','e','l','l','o')
```

```
function palyndrome($value as xs:string?) {  
  $value => characters()  
    => reverse()  
    => string-join()  
} as xs:string?
```

- **Fonctions sur les séquences**

- **fn:foot(seq) et fn:trunk(seq)**
  - Equivalent de **fn:head(seq)** et **fn:tail(seq)** mais sur les fins de séquence
- **fn:intersperse(seq, separator)**
  - Equivalent de **string-join(seq)** pour les séquences
- **fn:items-at(seq, at)**
  - Renvoie les items situés aux positions **at**
- **fn:replicate(seq, \$count)**
  - Produit **count** copies de la séquence
- **fn:slice(seq, start?, end?, step?)**
  - Construit une nouvelle séquence avec une tranche de la première

- **Fonctions sur les séquences**

- **fn:duplicate-values(seq, collation)**
  - Renvoie une sequence avec les valeurs qui apparaissent plus d'une fois
- **fn:starts-with-subsequence(seq, sub)**
- **fn:ends-with-subsequence(seq, sub)**
- **fn:contains-subsequence(seq, sub)**
  - Regarde si une séquence commence, se termine, ou contient une sous-séquence
- **fn:filter(seq, predicate(item, position))**
- **fn:every(seq, predicate(item, position))**
  - Vérifie que tous les items vérifient le prédicat

- **Fonctions sur les maps & arrays**
  - Identique dans l'esprit aux fonctions sur les séquences
  - Dans les namespace **map** et **array**

# Xslt 4.0

- **xsl:if @then @else**

- Deux nouveaux attributs optionnels

```
Hello <xsl:if test="$client.name" then="$client.name" else="World"/>!
```

- Pas de contenu si **@then** est présent

- **xsl:choose**

- les **xsl:when** et **xsl:otherwise** peuvent avoir un attribut **@select**

```
<xsl:template match="ol/li">
  <xsl:variable name="level" as="xs:number" select="count(ancestor::ol) mod 3"/>
  <xsl:variable name="list.style.type" as="xs:string">
    <xsl:choose>
      <xsl:when test="$level=1" select="'upper-alpha'"/>
      <xsl:when test="$level=2" select="'lower-greek'"/>
      <xsl:otherwise select="'decimal'"/>
    </xsl:choose>
  <xsl:variable>
    <ol style="list-style-type: {$list.style.type}">
      <xsl:apply-templates/>
    </ol>
  </xsl:template>
```

- Enfin !

```
<xsl:function name="f:days-in-month" as="xs:integer?">
  <xsl:param name="month-number" as="xs:integer"/>
  <xsl:param name="leap-year" as="xs:boolean"/>

  <xsl:switch select="$month-number">
    <xsl:when test="1, 3, 5, 7, 8, 10, 12" select="31"/>
    <xsl:when test="4, 6, 9, 11" select="30"/>
    <xsl:when test="2">
      <xsl:if test="$leap-year" then="29" else="28"/>
    </xsl:when>
  </xsl:switch>
</xsl:function>
```

- Paramètres optionnels

```
<xsl:function name="l:strangely-format" as="xs:string">

  <xsl:param name="text" as="xs:string"/>

  <xsl:param name="length.limit" as="xs:int" required="false" select="5"/>

  <xsl:param
    name="tete.trans" as="function(xs:string?) as xs:string"
    required="false"
    select="upper-case#1"/>

  <xsl:param
    name="queue.trans" as="function(xs:string?) as xs:string"
    required="false"
    select="lower-case#1"/>

  <xsl:sequence select="..." />
</xsl:function>
```

```
'abcdefghijklm' => l:strangely-format()                ⇒ 'AbcdeFghijKlm'
'abcdefghijklm' => l:strangely-format(queue.trans:=fn(s) {' '}) ⇒ 'A    F    K    '
'abcdefghijklm' => l:strangely-format(tete.trans :=fn(s) {' '}) ⇒ ' bcde ghij lm'
```



- **@as pour typer le retour du mode**
  - Permet de garantir la consistance des `xsl:template/@as`
- **xsl:mode peut contenir des templates**
  - Permet de ne pas spécifier le mode pour ces templates
  - Permet de regrouper les templates d'un même mode

```
<xsl:mode name="numbers-in-text" as="xs:string">

  <xsl:template match="type(xs:integer)[. gt 0 and . lt 21]"> ... </xsl:template>

  <xsl:template match="type(xs:integer)[. lt 10000]"> ... </xsl:template>

  <xsl:template match="type(xs:integer)"> ... </xsl:template>

</xsl:mode>
```

**Ce dont je n'ai pas parlé**

- **XDM**
  - <https://qt4cg.org/specifications/xpath-datamodel-40/Overview.html>
- **XQuery**
  - <https://qt4cg.org/specifications/xquery-40/xquery-40.html>
- **Serialization**
  - <https://qt4cg.org/specifications/xslt-xquery-serialization-40/Overview.html>
- **XPath language**
  - **for** généralisé aux **array**
- **XPath Function & Operators**
  - Fonctions de diagnostic, fonctions sur les nombres
  - **identity**, **void**, **atomic-equal**, **all-equal**, **all-different**
- **Xslt**
  - Fixed namespace, **xsl:item-type**

- **XML n'est pas mort !**

- En général 10 participants aux réunions hebdomadaires
- 20 personnes émettent des issues ⇒ lisent les specs...

- **Plusieurs implémentations des standards**

- Saxon
- BaseX
- Nouveau processeur Xslt par Mukul Gandhi

- **Objectif de sortir les standards pour 2025**

- Actuellement, la peinture n'est pas sèche...
- Beaucoup de changements à prévoir
- 141 issues ouvertes à ce jour
- Travaux sur XQuery 4.0 à peine entamés
- Aucun travaux sur sérialisation démarrés
- Certaines nouveautés non présentées ici

- **Merci de votre attention**

- Matériel de ce talk : <https://github.com/cmarchand/devoxx-2024>

