# Technologies XML

felipe@IFT3225 Hiver 2020

# Plan

- Déclarer une application XML
  - DTD (.dtd)
  - Schéma XML (.xsd)

- Feuilles de transformation
  - XSLT (.xsl)

# Facts

- XML = eXtensible Markup Language
- Largement inspiré de SGML (meta language de description de documents, IBM)
- Considéré par certains comme la plus grande avancée du Web
- Normalisé par W3C
  - XML 1.0 (défini en 1998)
  - XML 1.1
- Énormément d'applications XML
  - SVG, ATOM (standard), Music Markup Language, BeerXML, +
- A perdu du terrain comme format d'échange de données structurées (au profit de json), mais reste le langage de choix pour définir des formats de documents
- Ensemble de technologies:
  - XSL: langage de description de feuilles de style du W3C
    - XPath (sélection d'un noeud)
  - XSLT: feuilles de transformation

# Document bien formé versus valide

- **Bien formé** (un arbre)
  - une seule racine
  - les attributs ont une valeur qui est quêtée
  - <,>,& ne sont utilisés que dans leur contexte propre (à suivre)
  - débute avec l'entête
    <? xml version="1.0" encoding="utf-8" ?>
    facultatif

- **Valide** (pas n'importe quel arbre)
  - selon une définition (application)

% curl -o beer.xml http://www.beerxml.com/recipes.xml
% cat beer.xml

les éléments ou les noms d'attributs sont sensibles à la casse

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- BeerXML Format - Generated by BeerSmith  - see www.beersmith.com -->
<RECIPES>
<RECIPE>
 <NAME>Burton Ale</NAME>
 <VERSION>1</VERSION>
 <TYPE>All Grain</TYPE>
 <BREWER>Brad Smith</BREWER>
 <ASST_BREWER></ASST_BREWER>
 <BATCH_SIZE>18.92716800</BATCH_SIZE>
 <BOIL_SIZE>20.81988500</BOIL_SIZE>
 <BOIL_TIME>60</BOIL_TIME>
 <EFFICIENCY>72.0</EFFICIENCY>
 <HOPS>
<HOP>
 <NAME>Goldings, East Kent</NAME>
 <VERSION>1</VERSION>
 <ORIGIN>United Kingdom</ORIGIN>
 <ALPHA>5.50</ALPHA>
 <AMOUNT>0.0283500</AMOUNT>
 <USE>Boil</USE>
 <TIME>60.000</TIME>
 <NOTES>Used For: General purpose hops for bittering/finishing all British Ales
Aroma: Floral, aromatic, earthy, slightly sweet spicy flavor
Substitutes: Fuggles, BC Goldings
Examples: Bass Pale Ale, Fullers ESB, Samual Smith&amp;apos;s Pale Ale
</NOTES>
 <TYPE>Aroma</TYPE>
 <FORM>Pellet</FORM>
 <BETA>3.50</BETA>
 <HSI>35.0</HSI>
 <DISPLAY_AMOUNT>1.00 oz</DISPLAY_AMOUNT>
 <INVENTORY>1.00 oz</INVENTORY>
 <DISPLAY_TIME>60 min</DISPLAY_TIME>
</HOP>
...
```

une instance de l'application beerXML

# Tags versus attributs

- Il existe plusieurs façons d'encoder l'information
  - il y a un choix à faire entre encoder sous forme d'attribut ou sous forme de tag

For simplicity, the convention of using a separate tag for each data entry as in the following will be used:

```
<HOP>
 <NAME>Cascade</NAME>
…
</HOP>
```

- pro: extensible facilement (utile si le tag lui même est structuré)
- cons: verbeux

Though equivalent, the following XML format (i.e. XML Attributes) should **NOT** be used.

```
<HOP NAME="Cascade">
…
</HOP>
```

- pro: plus concis
- cons: pas extensible

Celui qui tranche est le concepteur de l'application (language) en spécifiant un **schéma**

# DTD (Document Type Definition)

- emprunté de SGML pour décrire la grammaire d'un document
- (sémantiquement) équivalent à BNF

**syntaxe**: <!keyword …>

racine

#quelconque de fois

facultatif

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE liste_de_gens [
 <!ELEMENT liste_de_gens (personne)*>
 <!ELEMENT personne (nom, date_de_naissance?, genre?, numero_de_secu?)>
 <!ELEMENT nom (#PCDATA)>
 <!ELEMENT date_de_naissance (#PCDATA)>
 <!ELEMENT genre (#PCDATA | masculin | féminin) "féminin">
 <!ELEMENT numero_de_secu (#PCDATA)>
]>
<liste_de_gens>
  <personne>
    <nom>Fred Bloggs</nom>
    <date_de_naissance>2008-11-27</date_de_naissance>
    <genre>masculin</genre>
  </personne>
</liste_de_gens>
```

la DTD

ELEMENT
ATTLIST
ENTITY
NOTATION

les deux plus fréquents

tout sauf <,>,&

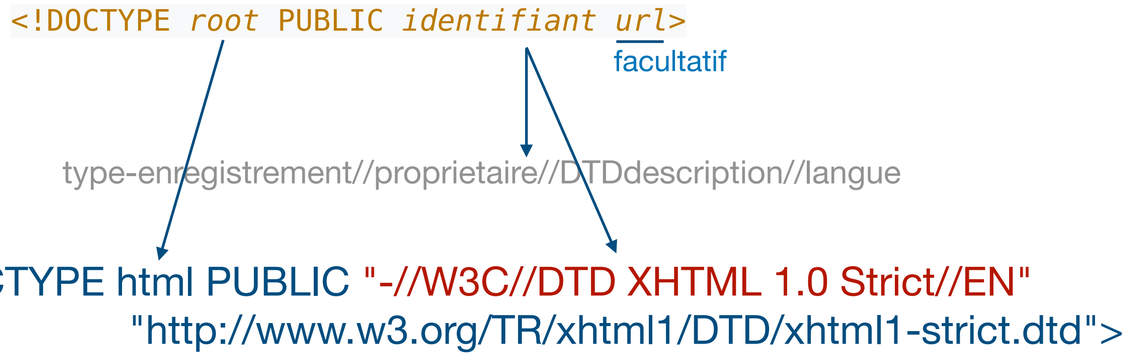
un document (ou instance) valide selon une DTD interne au document

# DTD externe

```
<?xml version="1.0"?>
<!DOCTYPE liste_de_gens SYSTEM url>
<liste_de_gens>
  <personne>
    <nom>Fred Bloggs</nom>
    <date_de_naissance>
      2008-11-27
    </date_de_naissance>
    <genre>masculin</genre>
  </personne>
</liste_de_gens>
```

Instance valide selon une DTD externe

Pour les DTD qui sont des normes il existe le mot **PUBLIC**




XHTML est une application XML définie par une DTD qui est une norme. L'URL est spécifiée pour information seulement

W3 https://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd

jugaison - Recherche Goo... | Page d'accueil IFT 1015 | Dan Tepfer: NPR Music Tiny Desk Con... | https:/

```
<!--
   Extensible HTML version 1.0 Strict DTD

   This is the same as HTML 4 Strict except for
   changes due to the differences between XML and SGML.

   Namespace = http://www.w3.org/1999/xhtml

   For further information, see: http://www.w3.org/TR/xhtml1

   Copyright (c) 1998-2002 W3C (MIT, INRIA, Keio),
   All Rights Reserved.

   This DTD module is identified by the PUBLIC and SYSTEM identifiers:

   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"

   $Revision: 1.1 $
   $Date: 2018/03/20 03:13:13 $

-->

<!--================= Character mnemonic entities =========================-->

<!ENTITY % HTMLlat1 PUBLIC
   "-//W3C//ENTITIES Latin 1 for XHTML//EN"
   "xhtml-lat1.ent">
%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC
   "-//W3C//ENTITIES Symbols for XHTML//EN"
   "xhtml-symbol.ent">
%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC
   "-//W3C//ENTITIES Special for XHTML//EN"
   "xhtml-special.ent">
%HTMLspecial;

<!--================== Imported Names ====================================-->

<!ENTITY % ContentType "CDATA">
    <!-- media type, as per [RFC2045] -->

<!ENTITY % ContentTypes "CDATA">
    <!-- comma-separated list of media types, as per [RFC2045] -->

<!ENTITY % Charset "CDATA">
    <!-- a character encoding, as per [RFC2045] -->

<!ENTITY % Charsets "CDATA">
    <!-- a space separated list of character encodings, as per [RFC2045] -->

<!ENTITY % LanguageCode "NMTOKEN">
    <!-- a language code, as per [RFC3066] -->

<!ENTITY % Character "CDATA">
    <!-- a single character, as per section 2.2 of [XML] -->
```

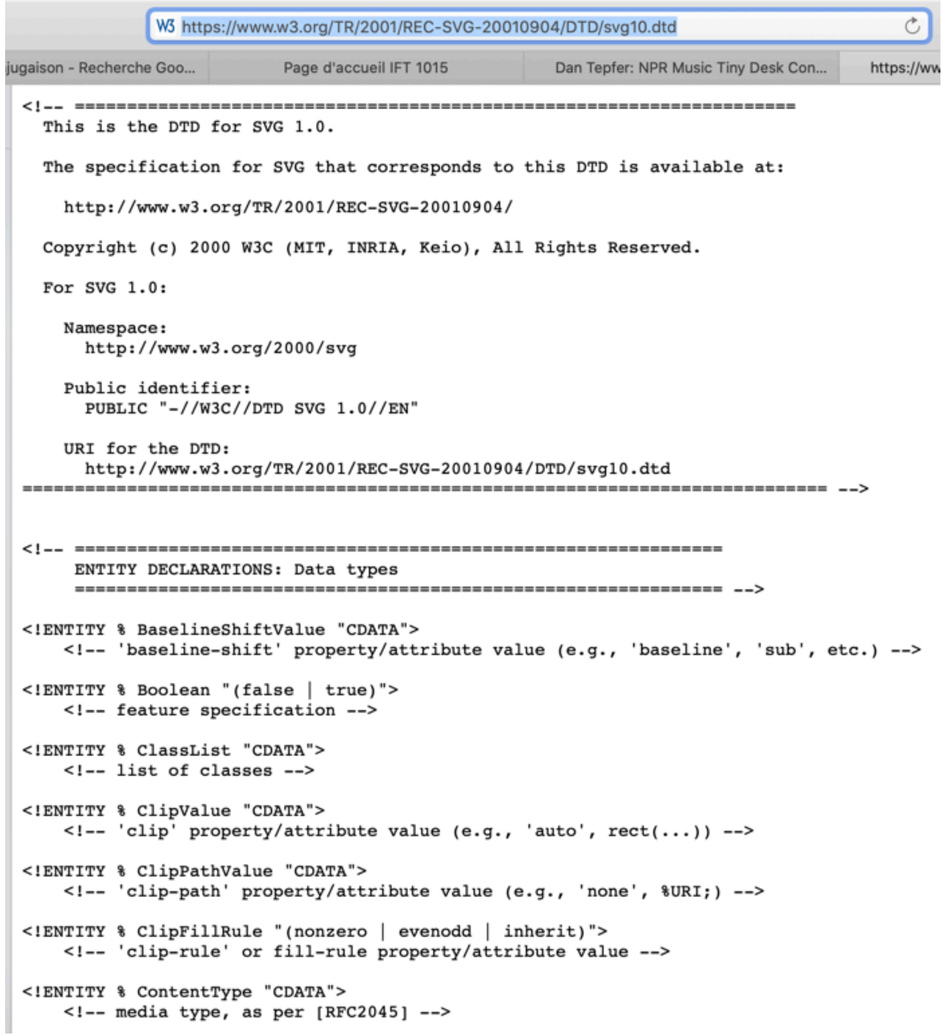
la meilleure façon d'apprendre les subtilités des DTDs …
et de XHTML1.0

← des macros

# SVG

## %curl https://svgsilh.com/svg/1299023.svg

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg version="1.0" xmlns="http://www.w3.org/2000/svg"
 width="1280.000000pt" height="1273.000000pt" viewBox="0 0 1280.000000 1273.000000"
 preserveAspectRatio="xMidYMid meet">
<metadata>
Created by potrace 1.15, written by Peter Selinger 2001-2017
</metadata>
<g transform="translate(0.000000,1273.000000) scale(0.100000,-0.100000)"
fill="#000000" stroke="none">
<path d="M9875 12720 c-142 -23 -234 -63 -338 -148 -93 -75 -249 -242 -303
-324 -55 -82 -86 -109 -183 -157 -101 -51 -687 -271 -721 -271 -7 0 -58 21
-114 47 -210 96 -325 123 -530 123 -142 0 -177 -10 -391 -115 -270 -132 -380
-212 -467 -339 -82 -120 -185 -375 -207 -511 -13 -86 -62 -187 -140 -292 -66
-89 -100 -124 -229 -241 -120 -109 -182 -201 -222 -334 -18 -55 -55 -153 -84
-217 -66 -144 -73 -167 -79 -262 -5 -75 -27 -147 -50 -161 -7 -4 -27 -8 -46
-8 -59 0 -108 -21 -178 -76 -123 -97 -176 -130 -313 -197 -74 -37 -160 -80
-190 -97 -60 -34 -72 -34 -170 -5 -96 29 -250 -4 -312 -66 -15 -15 -65 -45
-110 -68 -109 -53 -233 -135 -285 -188 -67 -70 -208 -362 -272 -569 -17 -54
-45 -166 -61 -249 -45 -224 -53 -246 -112 -312 -106 -120 -209 -303 -221 -389
-2 -23 -14 -55 -25 -70 -12 -16 -50 -103 -85 -194 -36 -91 -87 -218 -115 -282
-89 -203 -86 -184 -83 -668 l2 -425 -70 -38 c-39 -21 -112 -65 -162 -98 -94
-62 -133 -82 -142 -73 -8 8 23 90 86 234 66 152 107 276 141 435 72 330 108
  77…  -21 -53 52 87 145 261 175 26 4 50 8 53 9 3 0 30 -6 60 -15z"/>
<path d="M10253 12500 c-82 -29 -173 -99 -173 -133 0 -19 87 -47 148 -47 49 0
64 -4 69 -16 7 -18 6 -20 -54 -97 -49 -61 -61 -103 -40 -131 14 -19 17 -19
122 -4 90 14 114 22 144 46 118 95 130 213 32 324 -65 74 -148 94 -248 58z"/>
<path d="M7653 11440 c-60 -21 -103 -49 -133 -87 -46 -58 -17 -88 83 -86 40 1
80 0 90 -3 32 -10 18 -46 -43 -107 -33 -33 -60 -68 -60 -78 0 -29 49 -78 87
-88 109 -30 240 62 279 195 21 73 6 120 -64 195 -75 82 -136 97 -239 59z"/>
<path d="M7380 10946 c0 -14 -10 -33 -24 -43 -56 -44 -74 -138 -35 -186 26
-34 127 -82 214 -102 81 -19 137 -10 235 36 159 75 297 213 274 273 -10 26
-39 19 -145 -34 -212 -107 -302 -103 -433 19 -36 34 -70 61 -76 61 -5 0 -10
-11 -10 -24z"/>
…
</g>
</svg>
```




```
<!-- ================================================================
  This is the DTD for SVG 1.0.

  The specification for SVG that corresponds to this DTD is available at:

    http://www.w3.org/TR/2001/REC-SVG-20010904/

  Copyright (c) 2000 W3C (MIT, INRIA, Keio), All Rights Reserved.

  For SVG 1.0:

    Namespace:
      http://www.w3.org/2000/svg

    Public identifier:
      PUBLIC "-//W3C//DTD SVG 1.0//EN"

    URI for the DTD:
      http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd
================================================================== -->


<!-- ============================================================
     ENTITY DECLARATIONS: Data types
     ============================================================ -->

<!ENTITY % BaselineShiftValue "CDATA">
    <!-- 'baseline-shift' property/attribute value (e.g., 'baseline', 'sub', etc.) -->

<!ENTITY % Boolean "(false | true)">
    <!-- feature specification -->

<!ENTITY % ClassList "CDATA">
    <!-- list of classes -->

<!ENTITY % ClipValue "CDATA">
    <!-- 'clip' property/attribute value (e.g., 'auto', rect(...)) -->

<!ENTITY % ClipPathValue "CDATA">
    <!-- 'clip-path' property/attribute value (e.g., 'none', %URI;) -->

<!ENTITY % ClipFillRule "(nonzero | evenodd | inherit)">
    <!-- 'clip-rule' or fill-rule property/attribute value -->

<!ENTITY % ContentType "CDATA">
    <!-- media type, as per [RFC2045] -->
```

# Déclarer un élément

- un élément contenant des éléments: <!ELEMENT elt_name (list_of_childs)>
  - l'ordre des fils est important
  - des modificateurs permettent un peu de contrôle
    - +: au moins une fois
    - *: 0 ou plusieurs fois
    - ?: 0 ou 1

<!ELEMENT mail (from, to, date, body) >

- un élément sans fils <!ELEMENT elt_name categorie>
  - les catégories
    - EMPTY: element vide
    - PCDATA = Parsewd Character Data (chaine de caractères sauf <,>,&)
    - ANY contenu=texte ou élément de la DTD

<!ELEMENT mail (from, to+, date?, body) >

<!ELEMENT br EMPTY>
<!ELEMENT from (#PCDATA)>

- on peut également définir des **éléments mixtes**

<!ELEMENT root (#PCDATA|nom)*>
<!ELEMENT nom (#PCDATA)>

<root> bonjour</root>
<root>bonjour <nom>Léo</nom></root>
<root>bonjour <nom>Léo</nom><nom>Paul</nom> ça va ? </root>

sont des instances valides de root

# Déclarer les attributs

**2 syntaxes**

<!ATTLIST elt_name att_name att_type **default**>

<!ATTLIST elt_name
att_name1 att_type1 **default**1
att_name2 att_type2 **default**2
…>

- #REQUIRED
  - l'attribut est obligatoire
- #IMPLIED
  - l'attribut est optionnel
- #FIXED value
  - valeur (quotée) que tout élément aura comme valeur d'attribut
- value
  - valeur par défaut si attribut non spécifié

**xhtml1-strict.dtd**

macro

```
<!ELEMENT area EMPTY>
<!ATTLIST area
  %attrs;
  %focus;
  shape        %Shape;        "rect"
  coords       %Coords;       #IMPLIED
  href         %URI;          #IMPLIED
  nohref       (nohref)       #IMPLIED
  alt          %Text;         #REQUIRED
  >
```

<area>…</area> n'est pas valide => <area alt="…">…</area>

```
!ELEMENT html (head, body)>
<!ATTLIST html
  %i18n;
  id           ID             #IMPLIED
  xmlns        %URI;          #FIXED 'http://www.w3.org/1999/xhtml'
  >
```

<html xmlns="toto">…</html> n'est pas valide

# Déclarer les attributs

<!ATTLIST elt_name att_name **att_type** default>

- CDATA
  - tout sauf <,>,&
- (v1|v2|…vn)
  - énumération de valeurs possibles
- NMTOKEN
  - lettres, chiffres, ., -, _, :
- NMTOKENS = NMTOKEN + space
- ID
  - comme NMTOKEN mais commence par une lettre **et est unique dans le document**
- IDREF
  - ref à un ID existant dans le doc
- IDREFS
  - IREF séparés par des espaces

pour les principaux

**xhtml1-strict.dtd**

```
!ELEMENT html (head, body)>
<!ATTLIST html
  %i18n;
  id          ID              #IMPLIED
  xmlns       %URI;           #FIXED 'http://www.w3.org/1999/xhtml'
  >

<!ATTLIST label
  %attrs;
  for         IDREF           #IMPLIED


<!ATTLIST a
  %attrs;
  %focus;
  charset     %Charset;       #IMPLIED
  type        %ContentType;   #IMPLIED
  name        NMTOKEN         #IMPLIED
  href        %URI;           #IMPLIED

  avec    <!ENTITY % URI "CDATA">


 <!ATTLIST area
   %attrs;
   %focus;
   shape       %Shape;         "rect"

  avec   <!ENTITY % Shape "(rect|circle|poly|default)">
```

# Entités

- Entités générales
  - <!ENTITY nom "texte">

    **note**: macro pour l'instance

  - externes: <!ENTITY nom SYSTEM url>

    **note**: utile pour les contenus binaires

- Entités interne
  - <!ENTITY % nom text>

    **note**: macro à des fins internes à la DTD

toute instance XML a accès aux 5 entités suivantes:
- & &
- < <
- > >
- " "
- ' '

définition: <!ENTITY diro "@diro">
usage: venez au &diro; 

définition: <!ENTITY felipe SYSTEM "images/felipe.gif" NDATA GIF89a>
<!ATTLIST image source ENTITY #REQUIRED>
usage: <image source="felipe" />

autre type

```
<!ENTITY % events
 "onclick     %Script;       #IMPLIED
  ondblclick  %Script;       #IMPLIED
  onmousedown %Script;       #IMPLIED
  onmouseup   %Script;       #IMPLIED
  onmouseover %Script;       #IMPLIED
  onmousemove %Script;       #IMPLIED
  onmouseout  %Script;       #IMPLIED
  onkeypress  %Script;       #IMPLIED
  onkeydown   %Script;       #IMPLIED
  onkeyup     %Script;       #IMPLIED"
  >

<!ATTLIST map
  %i18n;
  %events;
  id          ID             #REQUIRED
```

# EXEMPLE

## book.dtd

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT book (toc?, content) >
<!ELEMENT content (chapter+, index?) >
<!ELEMENT toc (entry+) >

<!ELEMENT entry EMPTY>
<!ATTLIST entry type (chapter|index) #REQUIRED >
<!ATTLIST entry ref IDREF #REQUIRED >
<!ATTLIST entry page CDATA #IMPLIED >

<!ELEMENT chapter (title, text) >
<!ELEMENT title (#PCDATA) >
<!ATTLIST title id ID #REQUIRED >
<!ELEMENT text (#PCDATA) >

<!ELEMENT index (word+) >
<!ELEMENT word (#PCDATA) >
<!ATTLIST word pages NMTOKENS #REQUIRED >
```

## book.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- un exemple de DTD decrite en plusieurs parties (interne et externe)
    et d'instance decrite en plusieurs parties aussi
-->
<!DOCTYPE book SYSTEM "book.dtd"
[
<!ENTITY toc SYSTEM "chapters/toc.xml">
<!ENTITY chap1 SYSTEM "chapters/c1.xml">
<!ENTITY chap2 SYSTEM "chapters/c2.xml">
<!ENTITY index SYSTEM "chapters/index.xml">
]>
<book>
    &toc;
    <content>
        &chap1;
        &chap2;
        &index;
    </content>
</book>
```

## toc.xml

```
<toc>
    <entry type="chapter" ref="chap1" page="2"/>
    <entry type="chapter" ref="chap2"/>
</toc>
```

## index.xml

```
<index>
    <word pages="12 23">text</word>
    <word pages="7">word</word>
</index>
```

## c1.xml

```
<chapter>
    <title id="chap1">Titre du chapitre 1</title>
    <text>This is the text of chapter 1. </text>
</chapter>
```

## c2.xml

```
<chapter>
    <title id="chap2">Titre du chapitre 2</title>
    <text>This is the text of chapter 2. </text>
</chapter>
```

# Schéma XML

Même rôle que la DTD, mais en XML.
Offre également plus de contrôle sur les types.

préfixe associé au *namespace*

**cd-with-schema.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- les données proviennent de w3schools.com –>

<catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="cd.xsd">

        <cd code="BD-EB">
            <title tag="essai">Empire Burlesque</title>
            <artist>Bob Dylan</artist>
            <country>USA</country>
            <company>Columbia</company>
            <price>10.90</price>
            <year>1985</year>
        </cd>
        <cd code="BT-HYH">
            <title>Hide your heart</title>
            <artist>Bonnie Tyler</artist>
            <country>UK</country>
            <company>CBS Records</company>
            <price>9.90</price>
            <year>1988</year>
        </cd>
```

schéma XML

préfixe associé au *namespace*

le *namespace* des schémas XML

**cd.xsl**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- facon: poupées russes -->

<xs:element name="catalog">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="cd" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="title">
                            <xs:complexType>
                                <xs:simpleContent>
                                    <xs:extension base="xs:string">
                                        <xs:attribute name="tag" type="xs:string" />
                                    </xs:extension>
                                </xs:simpleContent>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="artist" type="xs:string" />
                        <xs:element name="country" type="xs:string" />
                        <xs:element name="company" type="xs:string" />
                        <xs:element name="price" type="xs:float" />
                        <xs:element name="year" type="xs:int" />
                    </xs:sequence>
                    <xs:attribute name="code" type="xs:string" />
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```

en bleu les mots connus du *namespace* des schémas

**syntaxe plus verbeuse ! (mais plus de contrôle)**

types

# Éléments simples

- Un élément **simple** contient seulement du texte (string, decimal, integer, boolean, date, time, …) ← types simples

préfixe

```
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:int" />
<xs:element name="naissance" type="xs:date" />
```

- On peut forcer la valeur d'un élément

```
<xs:element name="name" type="xs:string" fixed="toto"/>
```

- Un élément simple ne possède pas d'attribut
- Les éléments non simples sont de **type complexe**

```
<name>Jean</name>
<age>42</age>
<naissance>2019-10-09</naissance>
```

# Restriction d’un type simple

On peut **restreindre** un type simple à l’aide de **facettes**

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

un entier entre 0 et 120 (inclus)

<age>-15</pays> serait une erreur

```
<xs:element name="pays">
  <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:enumeration  value="Canada"/>
        <xs:enumeration  value="Japon"/>
        <xs:enumeration  value="Suède"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Canada|Japon|Suède

<pays>Autriche</pays> serait une erreur

clair mais très verbeux !

```
<xs:element name="x">
  <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z][a-z0-9]?"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

<x>aa0</x> serait une erreur

```
<xs:element name="secret">
  <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:minLength value="4"/>
        <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

<secret>aa0</secret> serait une erreur

# Les types complexes

- Tout élément non simple est **complexe**
  - Un élément qui contient d'autres éléments est **complexe**
  - Un élément avec des attributs est **complexe**
  - Un élément vide est **complexe** !

- Déclaration d'un élément contenant du texte et un attribut:

```
<xs:element name="x">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="tag" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

<x tag="bonjour">le monde</x> est valide

- Déclaration d'un élément vide avec attribut

```
<xs:element name="x">
  <xs:complexType>
        <xs:attribute name="tag" type="xs:string"/>
 </xs:complexType>
</xs:element>
```

- Déclaration d'un élément en contenant d'autres

```
<xs:element name="program">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="asa" type="_asa" minOccurs="1"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
        </xs:complexType>
</xs:element>
```

conteneur

défaut

```
<program name="mon_code">
  <asa>
   …
  </asa>
  <asa>
   …
  </asa>
</program>
```

conteneurs:

- **sequence**: ordre d'énumération important
- **all**: pas d'ordre, chaque élément doit arriver une fois
- **choice**: un élément parmi ceux décrits

on peut également contrôler le nombre de répétitions d'un élément (minOccurs, maxOccurs)

# On peut déclarer des types

Pour alléger la syntaxe du code en "poupée russe" et augmenter la réutilisabilité

```
<xs:simpleType name="_operator">
        <xs:restriction base="xs:string" >
            <xs:enumeration value="add" />
            <xs:enumeration value="sub" />
            <xs:enumeration value="mul" />
            <xs:enumeration value="div" />
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="_asa">
    <xs:sequence>
       <xs:element name="comment" type="xs:string" minOccurs="0"/>
       <xs:element name="tree" type="_node" />
    </xs:sequence>
</xs:complexType>

<!-- -->
<xs:simpleType name="_identifier">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z][a-z0-9]?"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="_entier">
    <xs:attribute name="value" type="xs:int" />
</xs:complexType>

<xs:complexType name="_let">
    <xs:sequence>
        <xs:element name="lvalue" type="_identifier" />
        <xs:element name="rvalue" type="_node" />
    </xs:sequence>
</xs:complexType>
```

```
<xs:element name="program">
       <xs:complexType>
            <xs:sequence>
                <xs:element name="asa" type="_asa" minOccurs="1"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
       </xs:complexType>
</xs:element>
```

**Il faut néanmoins être capable de distinguer un type simple d'un type complexe**

# Exemple

## code.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >

<!--  a code is a sequence of ASA -->
<xs:element name="program">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="asa" type="_asa" minOccurs="1"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
        </xs:complexType>
</xs:element>

<!-- an ASA is a comment and a node -->
<xs:complexType name="_asa">
    <xs:sequence>
       <xs:element name="comment" type="xs:string" minOccurs="0"/>
       <xs:element name="tree" type="_node" />
    </xs:sequence>
</xs:complexType>


<!-- un noeud -->
<xs:complexType name="_node">
    <xs:choice>
        <xs:element name="entier" type="_entier"/>
        <xs:element name="variable">
            <xs:complexType>
               <xs:attribute name="name" type="_identifier" use="required" />
            </xs:complexType>
        </xs:element>
        <xs:element name="let" type="_let" />
        <xs:element name="op" type="_binary_op" />
    </xs:choice>
</xs:complexType>
…
</xs:schema>
```

au moins un <asa>

une séquence d'un <comment> (qui peut être absent) et d'un <tree>

un noeud est soit <entier>, <variable>, <let> ou <op>

attribut obligatoire

**note**: les types définis n'ont pas à commencer par _ (juste une convention que j'ai adoptée ici)

## code-with-schema.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="code.xsd"
         name="Test">

    <!-- a code is a succession of asa -->

    <asa>
        <comment>an integer is evaluated as itself</comment>
        <tree>
            <entier value="12"/>
        </tree>
    </asa>

    <asa>
        <comment>a variable is (currently) not evaluated</comment>
        <tree>
            <variable name="ab"/>
        </tree>
    </asa>

    <asa>
        <comment> y = 12; </comment>
        <tree>
            <let>
                <lvalue>y</lvalue>
                <rvalue>
                    <entier value="12"/>
                </rvalue>
            </let>
        </tree>
    </asa>

…

</program>
```

# Exemple

## code.xsd

```
<xs:simpleType name="_operator">
        <xs:restriction base="xs:string" >
            <xs:enumeration value="add" />
            <xs:enumeration value="sub" />
            <xs:enumeration value="mul" />
            <xs:enumeration value="div" />
        </xs:restriction>
</xs:simpleType>


<!--  -->
<xs:simpleType name="_identifier">
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z][a-z0-9]?"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="_entier">
    <xs:attribute name="value" type="xs:int" />
</xs:complexType>

<xs:complexType name="_let">
    <xs:sequence>
        <xs:element name="lvalue" type="_identifier" />
        <xs:element name="rvalue" type="_node" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="_binary_op">
    <xs:sequence>
        <xs:element name="arg1" type="_node" />
        <xs:element name="arg2" type="_node" />
    </xs:sequence>
    <xs:attribute name="operator" type="_operator" use="required"/>
</xs:complexType>
```

une des 4 valeurs possibles

## code-with-schema.xml

```
<asa>
    <comment> une affectation plus grande a = (x * (y + 2)) - 4; </comment>
    <tree>
        <let>
            <lvalue>a</lvalue>
            <rvalue>
                <op operator="sub">
                    <arg1>
                        <op operator="mul">
                            <arg1>
                                <variable name="x"/>
                            </arg1>
                            <arg2>
                                <op operator="add">
                                    <arg1>
                                        <variable name="x"/>
                                    </arg1>
                                    <arg2>
                                        <entier value="2"/>
                                    </arg2>
                                </op>

                            </arg2>
                        </op>

                    </arg1>
                    <arg2>
                        <entier value="4"/>
                    </arg2>
                </op>

            </rvalue>
        </let>
    </tree>
</asa>
```

# Feuille de style (CSS)

On peut styler du XML avec du CSS, mais c'est un peu limité

**cd-withCSS.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd.css"?>

<!-- les données proviennent de w3schools.com -->

<catalog>

        <cd code="BD-EB">
            <title tag="essai">Empire Burlesque</title>
            <artist>Bob Dylan</artist>
            <country>USA</country>
            <company>Columbia</company>
            <price>10.90</price>
            <year>1985</year>
        </cd>
        <cd code="BT-HYH">
            <title>Hide your heart</title>
            <artist>Bonnie Tyler</artist>
            <country>UK</country>
            <company>CBS Records</company>
            <price>9.90</price>
            <year>1988</year>
        </cd>
```

**cd.css**

```
catalog { background-color: #fff; width: 100%; }

cd { display: block; margin-bottom: 30pt; margin-left: 0; }
title { background-color: #dc5; color: #FF0000; font-size: 20pt; }
artist { color: #0000FF; font-size: 20pt; }

country, price, year, company
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

# Feuille de transformation (XSLT)

- On peut transformer une instance XML en une sortie (typiquement du HTML) à l'aide d'une feuille de transformation


- XSLT est un langage (XML) fonctionnel (comme Scheme, mais avec des parenthèses terriblement longues et typées)
  - permet de faire des "grep" sur des bouts d'arbres (très utile)
  - syntaxe très lourde

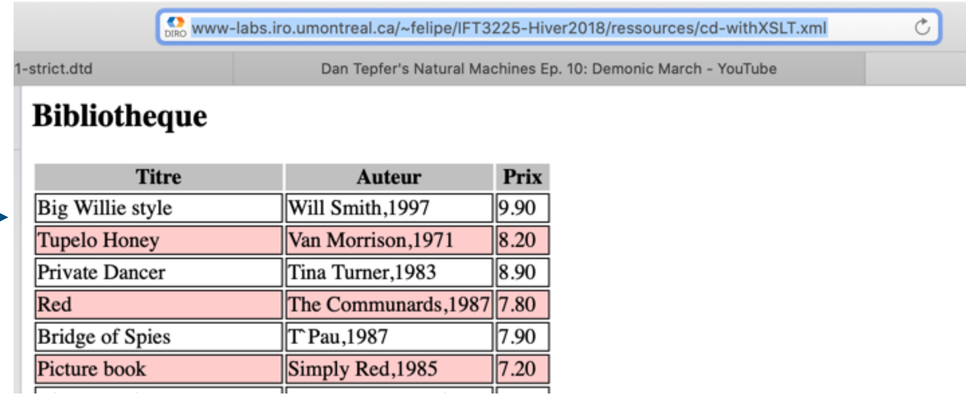
- Un éditeur comme oXygen (disponible gratuitement sur la logithèque de l'UdeM) est très utile (voire nécessaire).

cd-withXSLT.xml

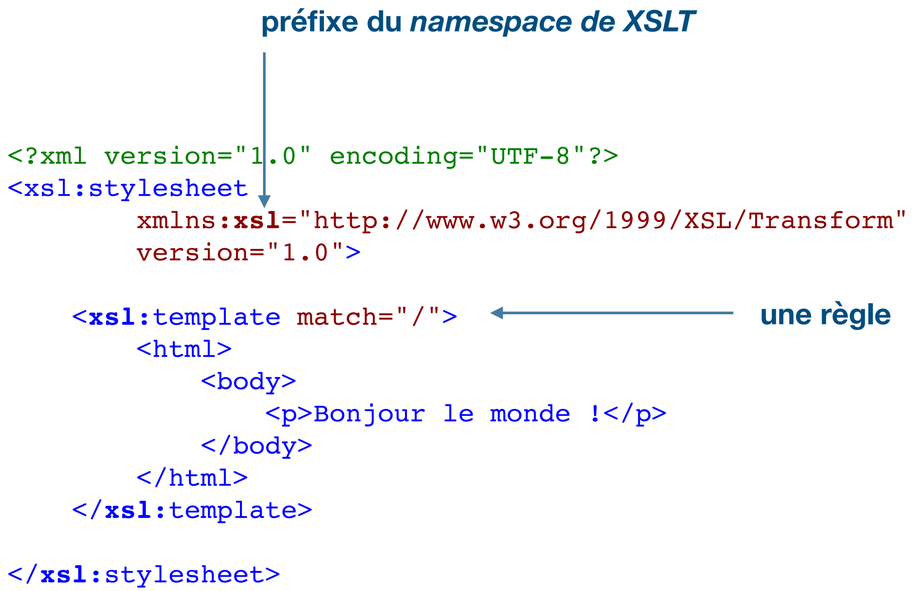
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="cd2.xsl"?>

<!-- les données proviennent de w3schools.com -->

<catalog>

        <cd code="BD-EB">
            <title tag="essai">Empire Burlesque</title>
            <artist>Bob Dylan</artist>
            <country>USA</country>
            <company>Columbia</company>
            <price>10.90</price>
            <year>1985</year>
        </cd>
…
</catalog>
```

lier l'instance à une feuille de transformation

le navigateur est capable de faire la transformation à la volée



# XSLT: Hello World




```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">

    <xsl:template match="/">
        <html>
            <body>
                <p>Bonjour le monde !</p>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

- Produit une page HTML (un paragraphe contenant le texte "Bonjour le monde !" et ce quelle que soit l'instance XML donnée en entrée (pas très utile).
- La règle indique sur quel noeud de l'instance *matcher (ici la racine) et le corps de la règle indique la transformation à effectuer (ici générer du HTML).*
- *Un programme XSLT contient au moins une règle (template).*

**Note**: à la lecture du XML, le navigateur applique la feuille de transformation à la volée et produit du HTML (Le code source est toujours le XML).

# XSLT & catalogue de CD

préfixe du *namespace de XSLT*

cd.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">

    <xsl:template match="/">
        <html>
            <body>
                <h2>My CD Collection</h2>
                <ul>
                    <xsl:for-each select="catalog/cd">
                        <xsl:sort select="artist" order="descending"/>
                        <li>
                            <xsl:value-of select="title"/> <br/>
                            <xsl:value-of select="artist"/>
                        </li>
                    </xsl:for-each>
                </ul>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

une règle

corps de la règle: code HTML (ou autre) avec des commandes XSLT

- la règle indique (via match) sur quel élément (de l'instance) elle s'applique
- le corps provoque (ici) la production de code HTML
- une boucle (xsl:for-each) permet d'indiquer quoi faire des fils du noeud *matché*
  - *affiche la valeur (xsl:value-of) des éléments <title> et <artist> de chaque élément <cd> de la collection*
  - *énumère les <cd> en ordre descendant du nom de l'artiste*

en prenant soin de changer dans cd-withXSLT.xml:

```
<?xml-stylesheet type="text/xsl" href="cd.xsl"?>
```

plus souple que CSS !

# XSLT & catalogue: variante

cd2.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl" exclude-result-prefixes="xd" version="1.0">
    <xd:doc scope="stylesheet">
        <xd:desc>
            <xd:p><xd:b>Created on:</xd:b> May 30, 2011</xd:p>
            <xd:p><xd:b>Author:</xd:b> felipe</xd:p>
            <xd:p/>
        </xd:desc>
    </xd:doc>
    <xsl:output method="html"/>

    <xsl:template match="/">
        <html>
            <head>
                <title>Ma CDtheque</title>
                <style type="text/css">
                    th {  background-color: silver;  }
                   .ligne {background-color: #FFCCCC; }
                    td {
                        border-style: solid;
                        border-width: 1px;
                    }</style>
            </head>
            <body>
                <h2>Bibliotheque</h2>
                <table>
                    <tr>
                        <th>Titre</th>
                        <th>Auteur</th>
                        <th>Prix</th>
                    </tr>
                    <xsl:for-each select="catalog/cd">
                        <xsl:sort select="artist" order="descending"/>
                        <xsl:sort select="year" order="ascending" data-type="number"/>
                        <tr>
                            <xsl:if test="position() mod 2 = 0">
                                <xsl:attribute name="class">ligne</xsl:attribute>
                            </xsl:if>

                            <td> <xsl:value-of select="title"/> </td>
                            <td> <xsl:value-of select="concat(artist, ',', year)"/> </td>
                            <td> <xsl:value-of select="price"/> </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

s'occupe de produire les entêtes. d'autres valeurs possibles (ex: `text`)

deux namespaces utilisés:

- xd (préfixe) est un espace de noms spécifique à oXygen permettant de renseigner l'output
- xsl (préfixe) est l'espace de noms du langage XSLT

**note**: le nom donné à un préfixe est au choix, on utilise habituellement un préfixe court. L'espace de noms est effectif dans l'élément qui le déclare (via attribut xmlns) et tous ses fils.
**note**: seuls les éléments du vocabulaire XSLT sont ici **qualifiés** (avec un préfixe). Le fait que les attributs (ex: `select`) et leurs valeurs connues (ex: `number`) ne le soient pas est lié à la façon dont le langage XSLT a été crée.

énumération en ordre descendant de valeur des éléments <artist> et à auteur égal, énumération selon l'année (ordre croissant).

instruction conditionnelle: ici, si la position du fils <cd> est paire (première position 1 selon la norme), alors mettre la valeur `ligne` à l'attribut `class` de l'élément <tr>

position et concat sont des fonctions XPath (ça vient…)

```
<catalog>

        <cd code="BD-EB">
            <title tag="essai">Empire Burlesque</title>
            <artist>Bob Dylan</artist>
            <country>USA</country>
            <company>Columbia</company>
            <price>10.90</price>
            <year>1985</year>
        </cd>
```

# XSLT & catalogue: variante++

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="catalog">
   <html>
      <head>
         <title> cd1.xsl </title>
      </head>
      <body>
         <h2> Application de cd1.xsl </h2>
         <xsl:apply-templates />
      </body>
   </html>
</xsl:template>

<xsl:template match="cd">
    <h4>CD</h4>
   <span style ="font-style: italic; color: blue;">Title:</span>
   <xsl:value-of select="title"/> <br/>

   <span style ="font-style: italic; color: red;">Artist:</span>
   <xsl:value-of select="artist"/> <br/>

   <span style ="font-style: italic; color: brown;">Country:</span>
   <xsl:value-of select="country"/> <br/>

</xsl:template>

</xsl:stylesheet>
```

2 règles

appliquer d'éventuels patrons (règles) sur les fils de <catalog> (donc les <cd>). Il est possible de filtrer les fils sur lesquels "relancer" (avec `select`)

# XSLT et factorielle

## factoriel.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="factoriel.xsl"?>

<factoriel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="factoriel.xsd">

    <fact>2</fact>
    <fact>3</fact>
    <fact>5</fact>
    <fact>10</fact>

    <fact>20</fact>
</factoriel>
```

## factoriel.xsd

juste pour la complétude, pas nécessaire au propos de XSLT !

```xml
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="factoriel">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="fact" type="xs:integer" minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

**vous rappelez-vous de votre première fonction récursive ?**

## factoriel.xsl

espace de nom par défaut
- utile si on veut que l'instance xsl soit valide.
- on pourrait associer un préfixe à l'espace de noms de xhtml mais il faudrait alors qualifier (préfixer) les éléments du language xhtml.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0"
                xmlns = "http://www.w3.org/1999/xhtml" >

<xsl:template match="factoriel">

   <html>
   <body>

   <h2>Output by factoriel.xsl</h2>

   <ul>
   <xsl:for-each select="fact">
      <li>
          fact(<xsl:value-of select="."/>) =
          <xsl:call-template name="fact"><xsl:with-param name="n" select="."/>
          </xsl:call-template>
      </li>
   </xsl:for-each>
   </ul>
   </body>
   </html>
 </xsl:template>

<!-- "fonction" factoriel -->
<xsl:template name="fact">
    <xsl:param name="n"/>

    <!– debug <xsl:message> n: <xsl:value-of select="$n"/> </xsl:message>-->

    <xsl:if test="$n = 1">
       1
    </xsl:if>

    <xsl:if test="$n > 1">
        <xsl:variable name="r">
            <xsl:call-template name="fact">
                <xsl:with-param name="n" select="$n - 1"/>
            </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$n * $r"/>
    </xsl:if>

</xsl:template>
</xsl:stylesheet>
```

template qui matche la racine

appel d'une règle comme une fonction

sorte de fonction avec un paramètre n

test d'arrêt

valeur de retour

récursivité

la variable r reçoit le résultat de l'appel au sous problème n-1

valeur de retour

# So far

Deux façons de faire des feuilles de transformation:

- une guidée par les données, via descente récursive  orchestrée par `apply-templates`
  - cas des CDs

- une guidée par les patrons (vu comme des fonctions) orchestrée par `call-template`
  - cas de factoriel

Une syntaxe un peu pénible, mais oXygen vous permet de ne presque rien taper.

XSLT Requiert de maîtriser:

- quelques structures de contrôle (comme dans tout language)
- XPath (ça vient)
- quelques concepts (introduits par l'exemple ici)

# XPath

## XPath

**XPath** est un langage de requête pour localiser une portion d'un document XML. Initialement créé pour fournir une syntaxe et une sémantique aux fonctions communes à XPointer et XSL, XPath a rapidement été adopté par les développeurs comme langage d'interrogation simple d'emploi.

syntaxe apparentée aux sélecteurs de CSS
avec des fonctions pour manipuler les parties sélectionnées

**sélecteurs**

| Expression | Description |
|---|---|
| *nodename* | Selects all nodes with the name "*nodename*" |
| / | Selects from the root node |
| // | Selects nodes in the document from the current node that match the selection no matter where they are |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects attributes |

| Path Expression | Result |
|---|---|
| /bookstore/book[1] | Selects the first book element that is the child of the bookstore element.<br><br>**Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath:<br><br>*In JavaScript:*<br>*xml*.setProperty("SelectionLanguage","XPath"); |
| /bookstore/book[last()] | Selects the last book element that is the child of the bookstore element |
| /bookstore/book[last()-1] | Selects the last but one book element that is the child of the bookstore element |
| /bookstore/book[position()<3] | Selects the first two book elements that are children of the bookstore element |
| //title[@lang] | Selects all the title elements that have an attribute named lang |
| //title[@lang='en'] | Selects all the title elements that have a "lang" attribute with a value of "en" |
| /bookstore/book[price>35.00] | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| /bookstore/book[price>35.00]/title | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

**note**: XPath en un slide est audacieux, mais il s'agit d'un langage relativement simple à prendre en main

### opérateurs

| Description | Example |
|---|---|
| Computes two node-sets | //book \| //cd |
| Addition | 6 + 4 |
| Subtraction | 6 - 4 |
| Multiplication | 6 * 4 |
| Division | 8 div 4 |
| Equal | price=9.80 |
| Not equal | price!=9.80 |
| Less than | price<9.80 |
| Less than or equal to | price<=9.80 |
| Greater than | price>9.80 |
| Greater than or equal to | price>=9.80 |
| or | price=9.80 or price=9.70 |
| and | price>9.00 and price<9.90 |
| Modulus (division remainder) | 5 mod 2 |

| Wildcard | Description |
|---|---|
| * | Matches any element node |
| @* | Matches any attribute node |
| node() | Matches any node of any kind |

# Un exemple plus complet

- "compilation" d'instances XML de code (code-with-schema.xml)
- par descente récursive
- transformation effectuée avec oXygen
    - Safari ne gère pas xPath2.0 (la feuille de transformation ne retourne rien)
    - fichiers copiés localement
        - l'instance: code-with-schema.xml
        - le schéma: code.xsd
        - la feuille de transformation: code.xsl

**note:** ma connaissance d'oXygen est limitée (il y a certainement des moyens plus élégants de faire ce qui suit)

# La "compilation"

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="code.xsl"?>

<program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="code.xsd"
         name="Test">

    <!-- a code is a succession of asa -->

    <asa>
        <comment>an integer is evaluated as itself</comment>
        <tree>
            <entier value="12"/>
        </tree>
    </asa>

    <asa>
        <comment>a variable is (currently) not evaluated</comment>
        <tree>
            <variable name="ab"/>
        </tree>
    </asa>

    <asa>
        <comment> y = 12; </comment>
        <tree>
            <let>
                <lvalue>y</lvalue>
                <rvalue>
                    <entier value="12"/>
                </rvalue>
            </let>
        </tree>
    </asa>

    <asa>
        <comment> une expression vaut elle-même </comment>
        <tree>
            <op operator="add">
                <arg1>
                    <entier value="2"/>
                </arg1>
                <arg2>
                    <entier value="3"/>
                </arg2>
            </op>
        </tree>
    </asa>

    <!-- une affectation -->
    <asa>
        <comment> affectation x = y + 3; </comment>
        <tree> ...
        </tree>
    </asa>

    <!-- une affectation plus importante -->
    <asa>
        <comment> une affectation plus grande a = (x * (y + 2)) - 4; </comment>
        <!-- comments are optional -->
        <tree>
            <let>
                <lvalue>a</lvalue>
                <rvalue>
                    <op operator="sub">
                        <arg1>
                            <op operator="mul">
                                <arg1>
                                    <variable name="x"/>
                                </arg1>
                                <arg2>
                                    <op operator="add">
                                        <arg1>
                                            <variable name="x"/>
                                        </arg1>
                                        <arg2>
                                            <entier value="2"/>
                                        </arg2>
                                    </op>

                                </arg2>
                            </op>

                        </arg1>
                        <arg2>
                            <entier value="4"/>
                        </arg2>
                    </op>

                </rvalue>
            </let>

        </tree>
    </asa>

</program>
```

## Output by program.xsl

**Program:** Test

**Asa: 1**
*// an integer is evaluated as itself*
*instruction:* 12 ;
*res:*12

**Asa: 2**
*// a variable is (currently) not evaluated*
*instruction:* ab ;

**Asa: 3**
*// y = 12;*
*instruction:* y := 12 ;

**Asa: 4**
*// une expression vaut elle-même*
*instruction:* add( 2 , 3 ) ;
*res:*5

**Asa: 5**
*// affectation x = y + 3;*
*instruction:* x := add( y , 3 ) ;

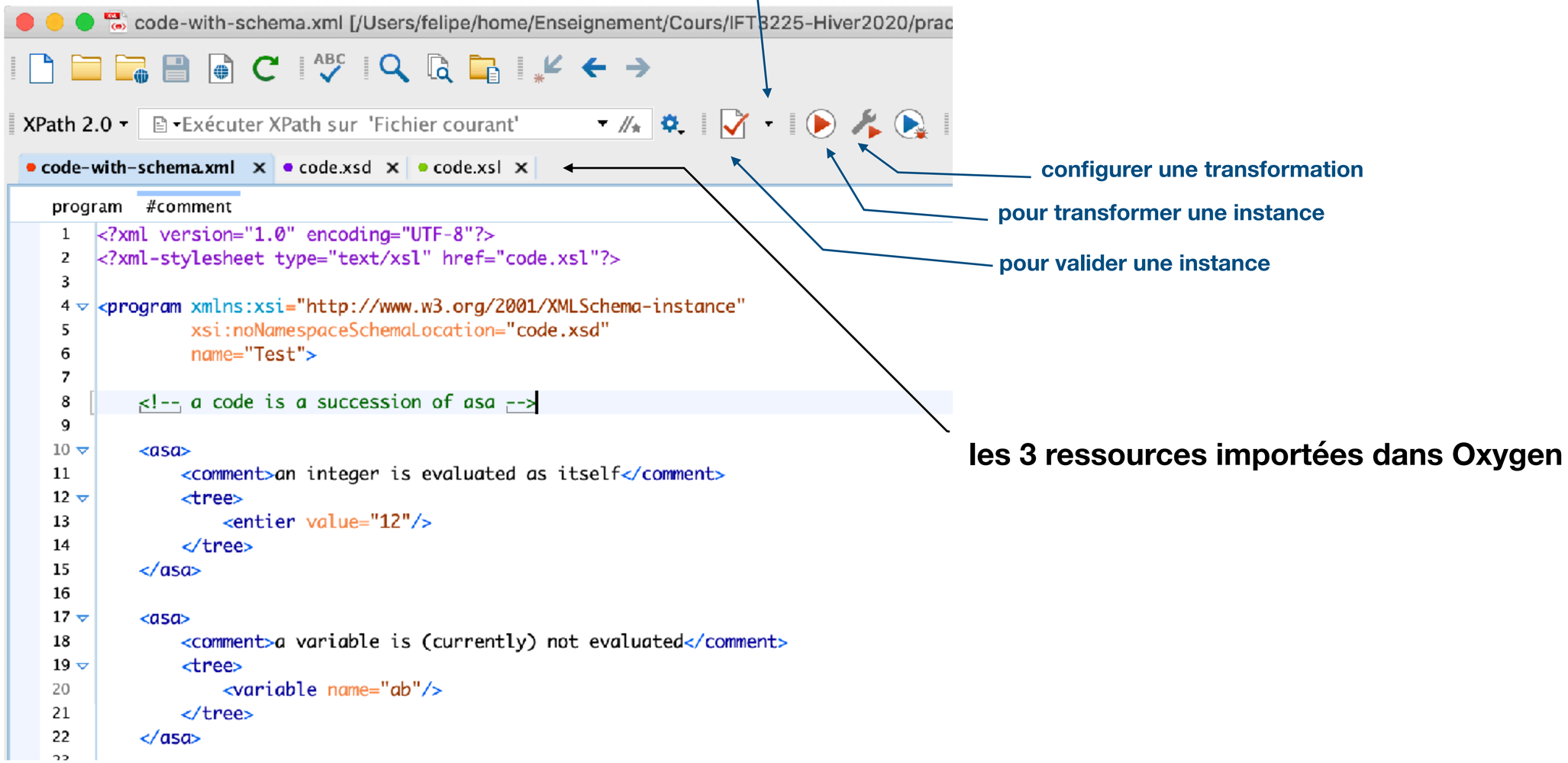
**Asa: 6**
*// une affectation plus grande a = (x * (y + 2)) - 4;*
*instruction:* a := sub( mul( x , add( x , 2 ) ) , 4 ) ;

**Asa: 7**
*// a big asa*
*instruction:* add( 2 , mul( sub( 4 , 1 ) , div( 4 , 3 ) ) ) ;
*res:*5

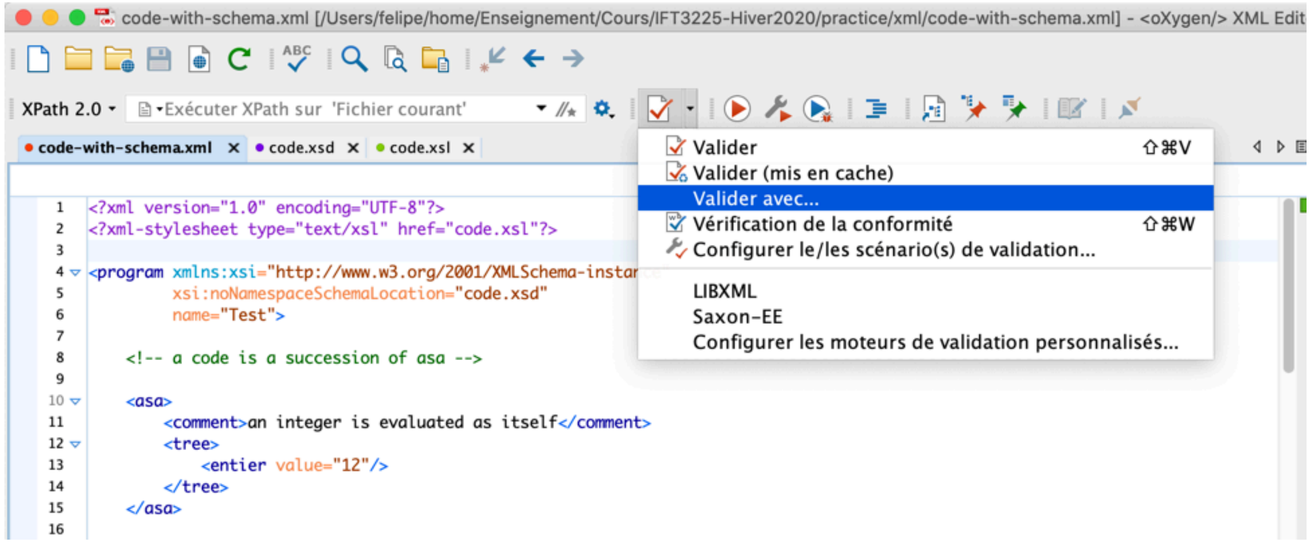
*variables utilisées:* ab y x
*lvalues:* y x a

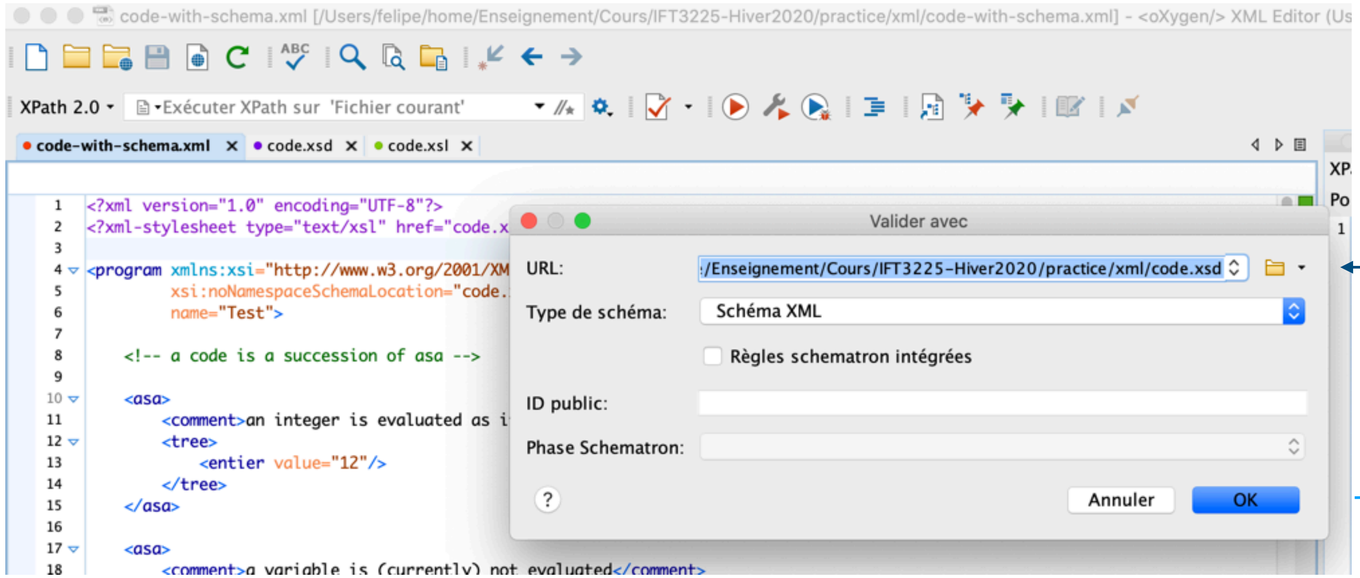
**lorsqu'il n'y a pas de variable dans l'ASA, le calcul est effectué**

# OXygen: prise en main



# Oxygen: valider

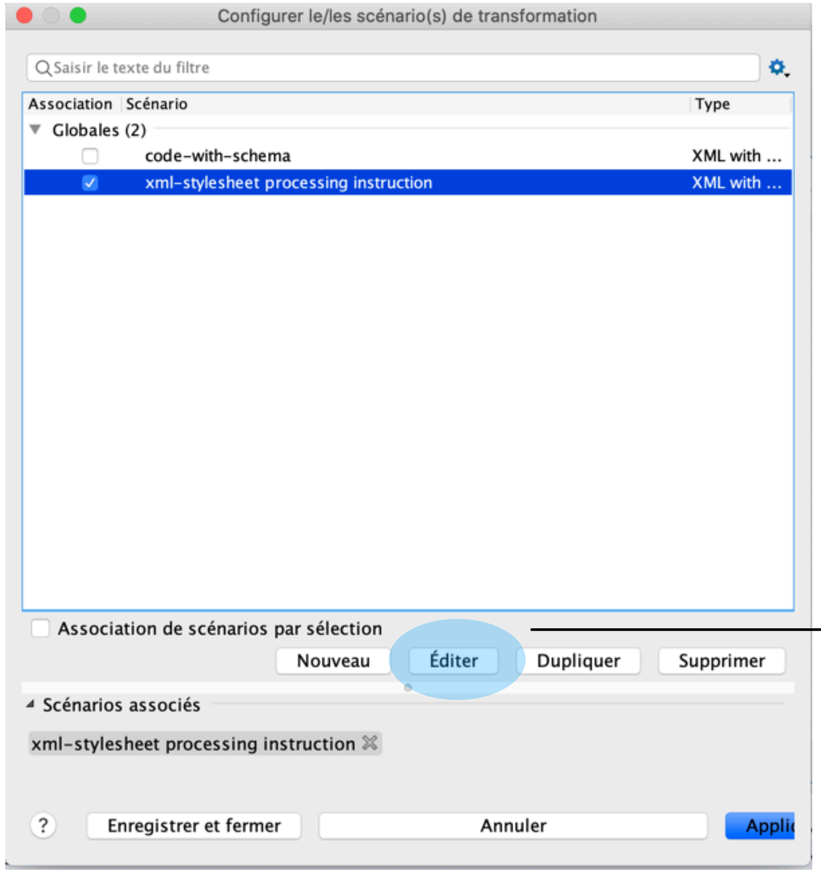


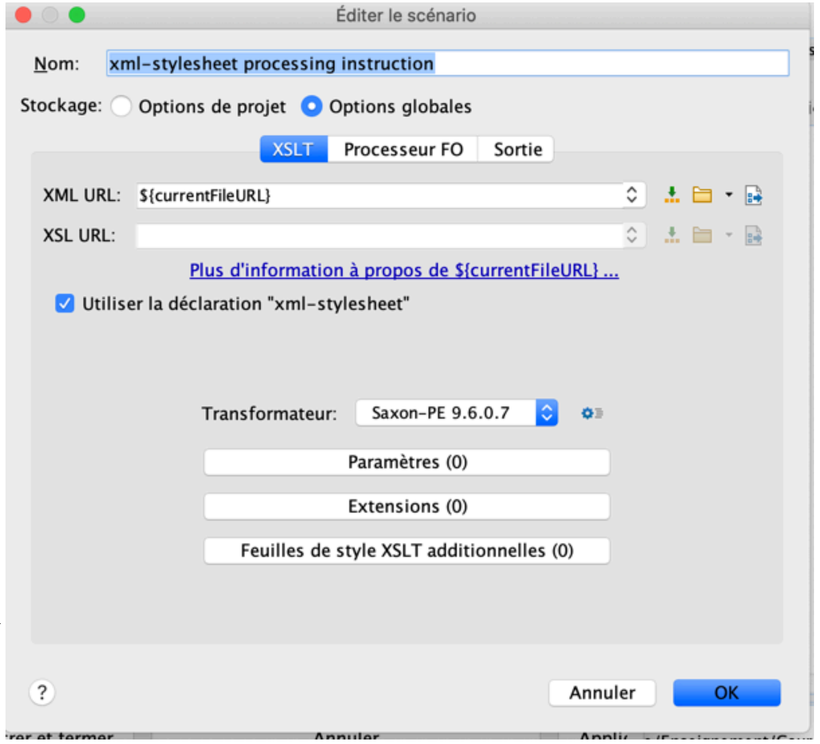





proposé directement car dans le même répertoire

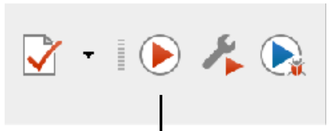
# Oxygen: transformer




**dit:** se fier aux déclarations de l'instance XML (qui se lie à code.xsl)

# Oxygen: transformer



code-with-schema.xml [/Users/felipe/home/Enseignement/Cours/IFT3225-Hiver2(

XPath 2.0 | Exécuter XPath sur 'Fichier courant'

code-with-schema.xml | code.xsd | code.xsl

```
2  <?xml-stylesheet type="text/xsl" href="code.xsl"?>
3
4  <program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5           xsi:noNamespaceSchemaLocation="code.xsd"
6           name="Test">
7
8      <!-- a code is a succession of asa -->
9
10     <asa>
11         <comment>an integer is evaluated as itself</comment>
12         <tree>
13             <entier value="12"/>
14         </tree>
15     </asa>
16
17     <asa>
18         <comment>a variable is (currently) not evaluated</comment>
19         <tree>
20             <variable name="ab"/>
21         </tree>
22     </asa>
23
24     <asa>
25         <comment> y = 12; </comment>
26         <tree>
```

Texte | Grille | Auteur

Description - 160 éléments

[Saxon-PE] name:Test

[Saxon-PE] calcul <tree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

## Output by program.xsl

**Program:** Test

**Asa: 1**
*// an integer is evaluated as itself*
*instruction:* 12 ;
*res:*12

**Asa: 2**
*// a variable is (currently) not evaluated*
*instruction:* ab ;

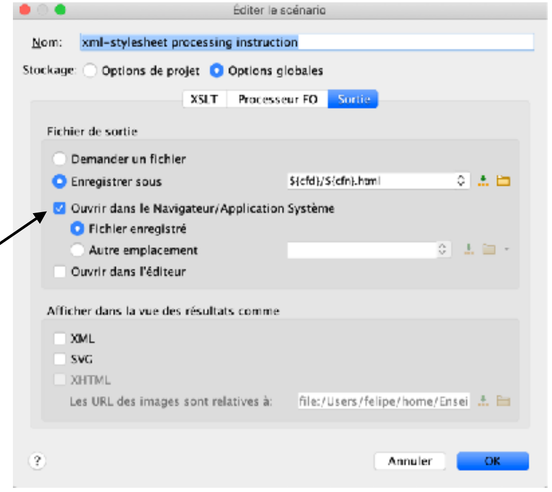
**Asa: 3**
*// y = 12;*
*instruction:* y := 12 ;

**Asa: 4**
*// une expression vaut elle-même*
*instruction:* add( 2 , 3 ) ;
*res:*5

**Asa: 5**
*// affectation x = y + 3;*
*instruction:* x := add( y , 3 ) ;

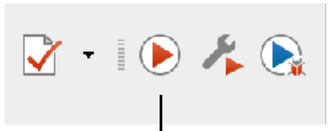
**Asa: 6**
*// une affectation plus grande a = (x * (y + 2)) - 4;*
*instruction:* a := sub( mul( x , add( x , 2 ) ) , 4 ) ;

**Asa: 7**
*// a big asa*
*instruction:* add( 2 , mul( sub( 4 , 1 ) , div( 4 , 3 ) ) ) ;
*res:*5




lance le navigateur par défaut pour afficher le résultat

# Oxygen: transformer



code-with-schema.xml [/Users/felipe/home/Enseignement/Cours/IFT3225-Hiver2(

XPath 2.0 | Exécuter XPath sur 'Fichier courant'

code-with-schema.xml | code.xsd | code.xsl

```
2  <?xml-stylesheet type="text/xsl" href="code.xsl"?>
3
4  <program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5           xsi:noNamespaceSchemaLocation="code.xsd"
6           name="Test">
7
8      <!-- a code is a succession of asa -->
9
10     <asa>
11         <comment>an integer is evaluated as itself</comment>
12         <tree>
13             <entier value="12"/>
14         </tree>
15     </asa>
16
17     <asa>
18         <comment>a variable is (currently) not evaluated</comment>
19         <tree>
20             <variable name="ab"/>
21         </tree>
22     </asa>
23
24     <asa>
25         <comment> y = 12; </comment>
26         <tree>
```

Texte | Grille | Auteur

Description - 160 éléments

[Saxon-PE] name:Test

[Saxon-PE] calcul <tree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

## Output by program.xsl

**Program:** Test

**Asa: 1**
*// an integer is evaluated as itself*
*instruction:* 12 ;
*res:*12

**Asa: 2**
*// a variable is (currently) not evaluated*
*instruction:* ab ;

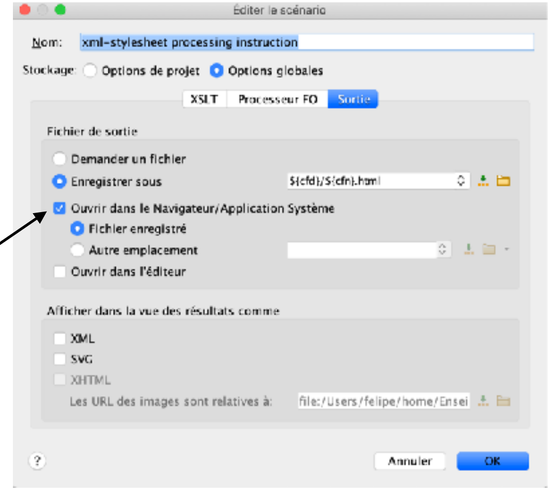
**Asa: 3**
*// y = 12;*
*instruction:* y := 12 ;

**Asa: 4**
*// une expression vaut elle-même*
*instruction:* add( 2 , 3 ) ;
*res:*5

**Asa: 5**
*// affectation x = y + 3;*
*instruction:* x := add( y , 3 ) ;

**Asa: 6**
*// une affectation plus grande a = (x * (y + 2)) - 4;*
*instruction:* a := sub( mul( x , add( x , 2 ) ) , 4 ) ;

**Asa: 7**
*// a big asa*
*instruction:* add( 2 , mul( sub( 4 , 1 ) , div( 4 , 3 ) ) ) ;
*res:*5

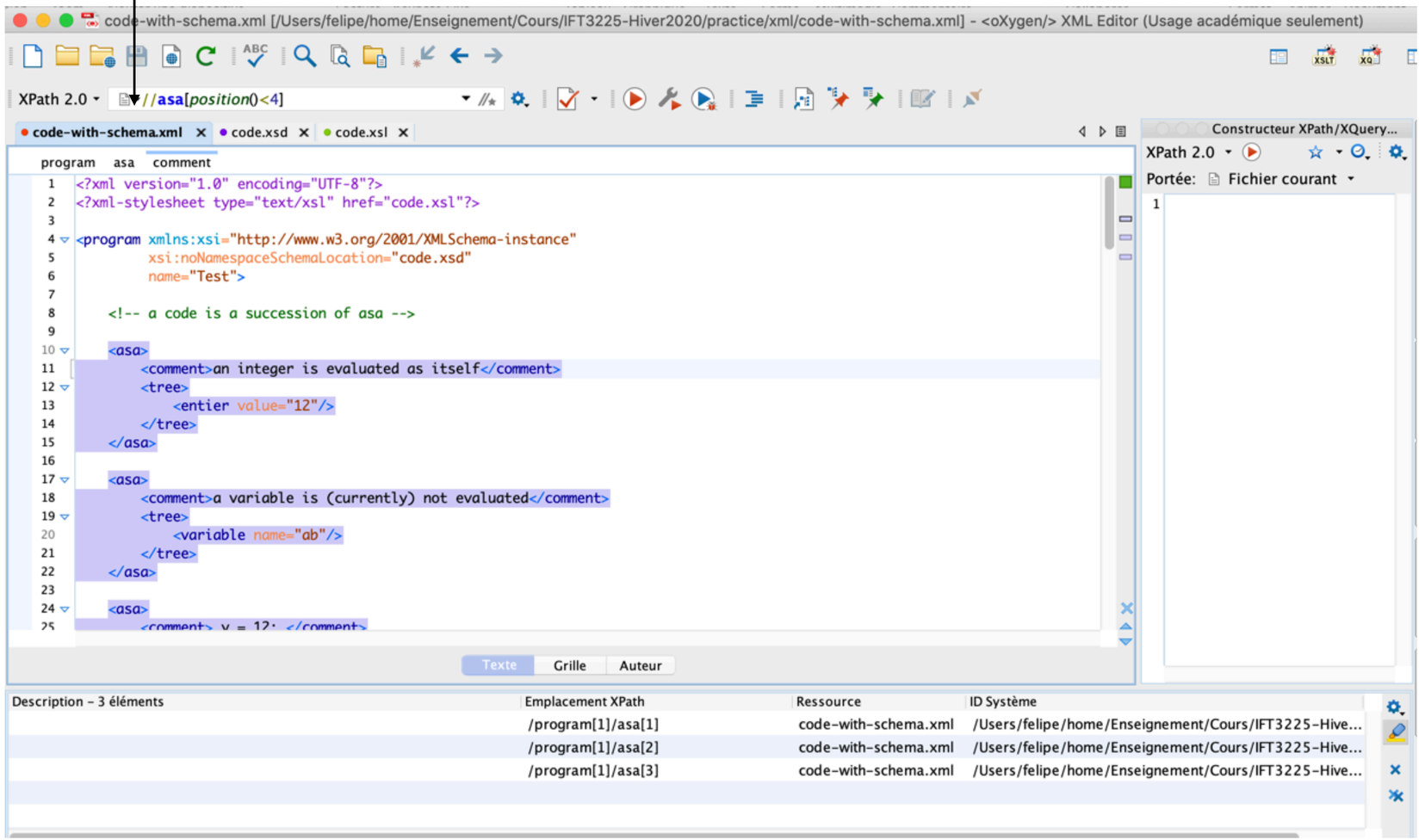



lance le navigateur par défaut pour afficher le résultat

# Oxygen & XPath

requête

résultat



# Quelques points sur `code.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns = "http://www.w3.org/1999/xhtml"
                version="2.0"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:pl="felipe">

<xsl:variable name="space"> <xsl:text> </xsl:text> </xsl:variable>
```

**espace de noms par défaut**

**espace de noms personnel**

**une "variable" (un noeud XML) dont le contenu est une espace**

# Quelques points sur `code.xsl`

```
<xsl:template match="program">
    <html>
    <body>
      <b>Program:</b> <xsl:value-of select="$space"/>

      <span class="progname"> <xsl:value-of select="@name"/> <br/> </span> <br/>

      <!-- a recursive call would do -->
      <xsl:for-each select="asa">

         <span class= "asa"> Asa:  <xsl:value-of select="position()" />  </span> <br/>
         <xsl:if test="count(comment) > 0">
            <span class="comment"> // <xsl:value-of select="comment" /> </span> <br />
         </xsl:if>

         <xsl:apply-templates select="tree" />
         <br/>
      </xsl:for-each>

      <!-- variables de programme analyse -->
      <em> variables utilisées: </em>
       <xsl:for-each select="distinct-values(//variable/@name)">
          <xsl:value-of select="."/>
          <xsl:value-of select="$space"/>
       </xsl:for-each>
       <br />

       <em>lvalues: </em>
         <xsl:for-each select="//lvalue">
            <xsl:value-of select="."/>
            <xsl:value-of select="$space"/>
         </xsl:for-each>
       <br />

      </body>
    </html>
</xsl:template>
```

valeur de l'attribut name

si un commentaire est présent

descente sur les fils <tree> (tous)

les valeurs des attributs name des éléments <variable> sans duplicatas

l'attribut en cours

# Quelques points sur `code.xsl`

```
<!-- node specific templates -->
<xsl:template match="tree">

  <em>instruction: </em> <xsl:apply-templates /> ;
  <br />

  <!-- si le noeud est calculable (que des valeurs) alors faire le calcul -->
  <xsl:if test="pl:isPlain(./*[1]) = true()">
    <em>res:</em>
    <xsl:call-template name="calcul">
      <xsl:with-param name="node" select="."/>
    </xsl:call-template>
    <br/>
  </xsl:if>

  <!-- j'aurais pu appeler un template:
      <xsl:call-template name="isPlain">
        <xsl:with-param name="node" select="."/>
      </xsl:call-template>
      <br />
  -->

</xsl:template>
```

**isPlain est un prédicat que j'ai crée qui retourne true si node ne contient pas de variable (descente récursive)**

```
<xsl:function name="pl:isPlain">
  <xsl:param name="node"/>

<!--  notez que message peut etre utilise pour debugger
  <xsl:message>
          isPlain <xsl:copy-of select="$node"/>
          <xsl:value-of select="local-name($node)"/>
  </xsl:message>
-->

  <xsl:choose>
    <xsl:when test="local-name($node) = 'entier'">
      <xsl:value-of select="true()"/>
    </xsl:when>
    <xsl:when test="local-name($node) = 'variable'">
      <xsl:value-of select="false()"/>
    </xsl:when>
    <xsl:when test="local-name($node) = 'let'">
      <xsl:value-of select="false()"/>
    </xsl:when>
    <xsl:when test="local-name($node) = 'op'">
      <xsl:variable name="r1" select="pl:isPlain($node/*[1]/*[1])" as="xs:boolean"/>
      <xsl:variable name="r2" select="pl:isPlain($node/*[2]/*[1])" as="xs:boolean"/>
      <xsl:value-of select="$r1 and $r2"/>
    </xsl:when>
  </xsl:choose>

</xsl:function>
```

appelé si aucune variable

variable non qualifiée

fils gauche

fils droit

2 "variables"

```
<xsl:template name="calcul">
  <xsl:param name="node"/>

  <xsl:choose>

    <xsl:when test="local-name($node) = 'tree'">
      <xsl:call-template name="calcul"><xsl:with-param name="node" select="$node/*[1]"/></xsl:call-template>
    </xsl:when>

    <xsl:when test="local-name($node) = 'entier'">
      <xsl:value-of select="$node/@value"/>
    </xsl:when>

    <xsl:when test="local-name($node) = 'op'">

      <xsl:variable name="r1">
        <xsl:call-template name="calcul"> <xsl:with-param name="node" select="$node/*[1]/*[1]"/></xsl:call-template>
      </xsl:variable>
      <xsl:variable name="r2">
        <xsl:call-template name="calcul"> <xsl:with-param name="node" select="$node/*[2]/*[1]"/></xsl:call-template>
      </xsl:variable>

      <xsl:choose>
        <xsl:when test="$node/@operator = 'add'">
          <xsl:value-of select="$r1 + $r2"/>
        </xsl:when>
        <xsl:when test="$node/@operator = 'sub'">
          <xsl:value-of select="$r1 - $r2"/>
        </xsl:when>
        <xsl:when test="$node/@operator = 'mul'">
          <xsl:value-of select="$r1 * $r2"/>
        </xsl:when>
        <xsl:when test="$node/@operator = 'div'">
          <xsl:value-of select="$r1 idiv $r2"/>
        </xsl:when>

      </xsl:choose>

    </xsl:when>
  </xsl:choose>

</xsl:template>
```

# code.xsl (suite & fin)

```
<xsl:template match="entier">
  <xsl:value-of select="@value"/>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="variable">
  <xsl:value-of select="@name"/>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="let">
  <xsl:value-of select="lvalue"/> :=
  <xsl:apply-templates select="rvalue" />
</xsl:template>

<xsl:template match="op">
  <xsl:value-of select="@operator"/>(
  <xsl:apply-templates select="arg1" />,
  <xsl:apply-templates select="arg2" />)
</xsl:template>
```

**les autres templates atteints par descente récursive**

**sélection des fils sur lesquels faire la descente**