

Technique et chaîne de publication électronique avec XSLT

Jean-Damien Généro

2023, 5 dec. - 2024, 23 janv.

École nationale des chartes – M2 TNAH

Contact

- jean-damien.genero@cnrs.fr

Objectifs

- XPath

- Naviguer dans un arbre XML
- Manipuler les principales fonctions XPath

- XSLT

- Manipuler les règles (*templates*) basiques
- Manipuler les conditions et les boucles

- Édition numérique

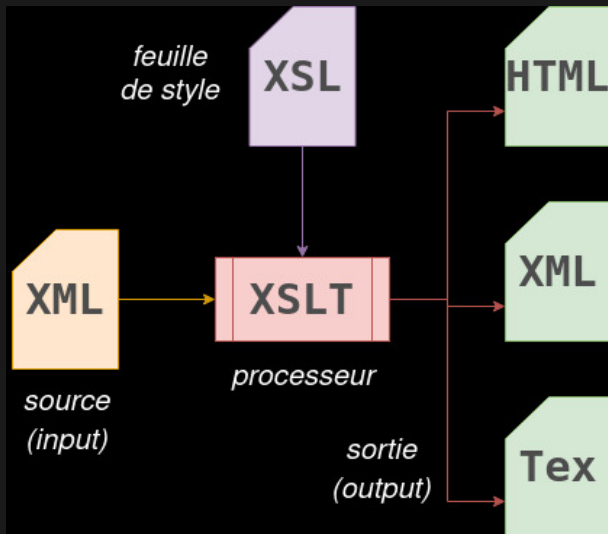
- Transformer un document XML en un autre document XML
- Transformer un document XML en un document HTML
- Transformer un document XML en un document \LaTeX

Introduction. Écosystème XML

Écosystème XML/ Principes généraux

- **XML** → *langage de balisage qui encode une description de la mise en page et de la structure logique d'un document*
 - **XPath** → *langage de requête pour naviguer dans la structure hiérarchique d'un document XML à l'aide de chemins;*
- **XSL** → *spécifications pour écrire des feuilles de style (stylesheet);*
- **XSLT** → *un langage conçu pour transformer des documents XML en d'autres documents selon les spécifications XSL.*

Écosystème XML/ Schéma



Écosystème XML/ Navigation XPath et de règles XSL

- **Navigation XPath :**
 - Doc XML = structuré, on parle d'« **arbre** » (*XML tree*);
 - **Itinéraire** vers une balise (« **parent** » → « **enfant** »);
 - Rédaction dans une **syntaxe propre**.
- **Règle XSL (*template*) :**
 - **Agir** sur une ou plusieurs balises (*transform*);
 - Exemples : **copier** balise + contenu, copier uniquement le contenu et l'**insérer** dans une nouvelle balise, **ajouter** ou **supprimer** un attribut, etc.;
 - Rédaction dans une **syntaxe XML** (langage à balises).
- **Utilisation avec d'autres langages**
(*Python : lxml*).

Écosystème XML/ Exemple d'un chemin XPath dans un XML tree

- Comment accéder aux balises `<p>` avec XPath ?

```
<TEI>
  <teiHeader/>
  <text>
    <body>
      <div>
        <p>title</p>
        <p>text</p>
      </div>
    </body>
  </text>
</TEI>
```

- `<TEI>` → `<text>` → `<body>` → `<div>` → `<p>`
- Traduction XPath : `/TEI/text/body/div/p`
- Simplification : `//body/div/p` (`/TEI/text/body/div/p`)

Écosystème XML/ Exemple d'une règle XSL

- Comment appliquer une règle XSL aux balises `<p>`?
 - Une règle XSL est toujours contenue dans une balise `<xsl:template/>` possédant un `@match`.
- Indiquer le chemin XPath vers les `<p>` dans l'`@match`.

```
<xsl:template match="/TEI/text/body/p">  
  <xsl:copy-of select="."/>  
</xsl:template>
```

ou

```
<xsl:template match="//body/p">  
  <xsl:copy-of select="."/>  
</xsl:template>
```

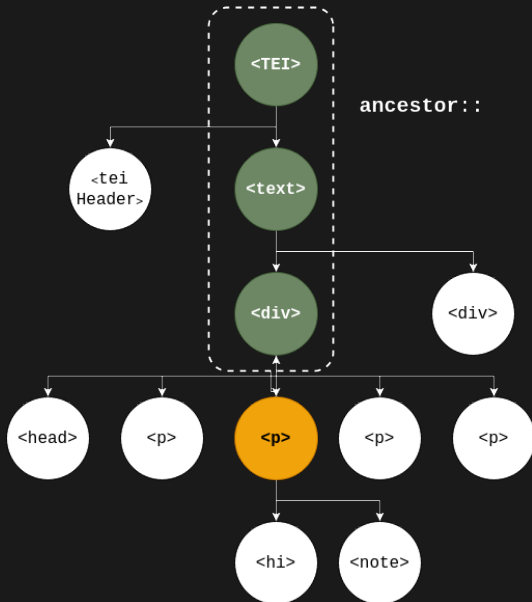
1. XPath

- **Nœud** (*node*) (7) = composant de l'arbre : *root*, *element*, *attribute*, *text*, *comment*, *namespace* et *processing instruction*.
- Deux rôles : *current node* (fixe, le premier du chemin) ou *context node* (variable, celui que XPath évalue à l'instant *t*);
- Écrire un chemin (*expression*) : succession de nœuds séparés par l'**opérateur** / ;
 - Tester dans Oxygen :
/TEI/text/body/div/p

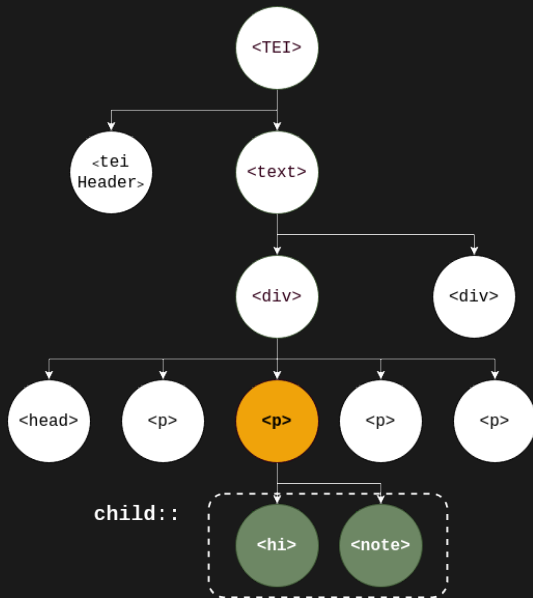
XPath/ Axes de relation

- Un axe permet de qualifier la **relation** entre le **nœud de contexte** et un ou des **autre(s) nœud(s)**.
- Direction basique : **parent** → **enfant**.
- 13 axes XPath, dont :
 - *parent::* → nœud immédiatement au-dessus;
 - *child::* → nœud immédiatement en-dessous;
 - *following-sibling::* → nœud(s) après celui de contexte et qui ne fait/ont pas partie de ses descendants;
 - *ancestor, descendant, preceding*, etc.

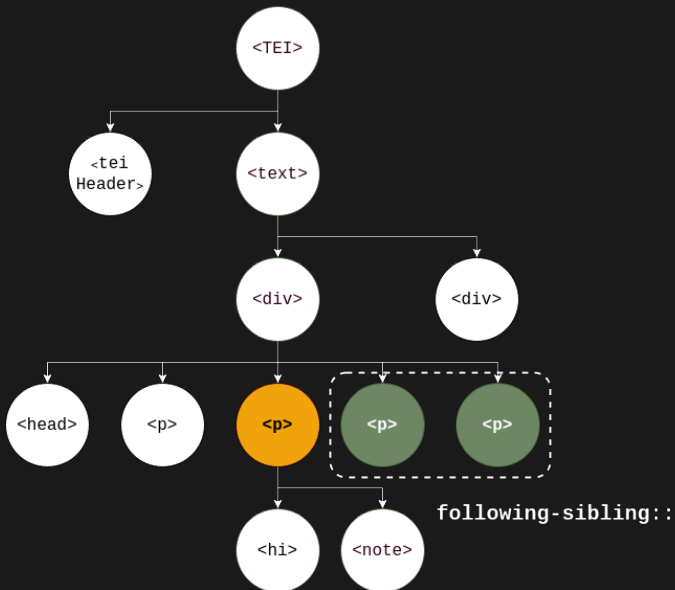
XPath/ Axe ancestor::



XPath/ Axe child::



XPath/ Axe following-sibling::



XPath/ Abréviations des axes

- 1^{er} élément de l'arbre (*root node*) → */*
(attention : différent de la racine <TEI>)
- *descendant-or-self::* → *//*
- *self::* → *.*
- *attribute::* → *@*
- Donc : vous pouvez écrire *//div/@n* au lieu de *//div/attribute::n*.

XPath/ Prédicats : définition

- Prédicat (filtre) = **condition** qui doit être satisfaite par le nœud de contexte (existence d'un attribut, valeur d'un attribut, position d'une balise, etc.).
- Écrit entre crochets `[]`.
- `//div[@n='2']/head` = le `<head>` de la `<div>` avec un `@n` de valeur 2.
- `//body/div[@n='2']/p[2] = ?`

XPath/ Prédicats : exemples

- `//tag[position()=2]` ou `//tag[2]` → condition de position (<tag> n°2);
- `//tag[last()]` → dernier <tag>;
- `//tag[@foo='bar']` → expression logique (<tag> avec un *@foo* dont la valeur est *bar*);

XPath/ Notions essentielles

- Arbre XML, racine, nœuds (7, notés avec ());
- Chemin XPath, opérateur /;
- Axes de relation (13, notés avec ::);
- Prédicats (notés avec []).

2. XSLT : élément racine et instructions de premier niveau

XSLT/ Définition

- XSLT → un **langage de programmation**;
- Permet de **transformer un doc XML** en un autre doc (*.xml*, *.html*, *.tex*, etc.);
- Transformation opérée par un **processeur XSLT**
 - **Construit** l'arbre output, **transforme** l'arbre input et **séréalise** le document output;
 - Le plus connu → **Saxon** (en ligne de commande);
 - Généralement **intégré** à des logiciels (**Oxygen**) ou des librairies (**python : lxml**).

XSLT/ La feuille de style

- Feuille de style XSL == un doc XML `.xsl`;
 - Contient des instructions ou règles (*templates*);
 - Élément racine : `<xsl:stylesheet>`
-
- Dans Oxygen → ouvrir un nouveau doc « XSLT Stylesheet ».
 - Observer les attributs de l'élément racine.

XSLT/ Élément racine : `<xsl:stylesheet>` (1/4)

Espace de noms et attributs présents de base dans Oxygen :

- *`@xmlns:xsl`* : espace de noms (namespace) XSL;
- *`@xmlns:xs`* : espace de noms XML;
- *`@exclude-result-prefixes`* : liste des préfixes qui ne seront pas copiés dans l'output;
- *`@version`* : version de XSL (1, 2 ou 3).

XSLT/ Élément racine : `<xsl:stylesheet>` (2/4)

Attributs à remplacer dans Oxygen *pour une transformation vers XML-TEI* :

- remplacer `@xmlns:xs` par `@xmlns:tei` → espace de noms de l'élément racine de l'output.
 - évite l'ajout de `tei:` à chaque élément "matché" par une règle (`@match`).
- remplacer `xs` par `tei` dans `@exclude-result-prefixes`.

XSLT/ Élément racine : `<xsl:stylesheet>` (3/4)

Attributs à ajouter dans Oxygen *pour une transformation vers XML-TEI* :

- `@xpath-default-namespace` : espace de noms des chemins XPath de la feuille de style (<http://www.tei-c.org/ns/1.0>).
 - évite de devoir écrire le préfixe `tei:` devant chaque nœud d'une expression XPath.
- `@xmlns` avec l'adresse de l'espace de noms TEI.
 - détermine l'espace de noms de l'ensemble du document de sortie.

XSLT/ Élément racine : `<xsl:stylesheet>` (4/4)

- **attention** : TEI est un standard XML parmi d'autres ! Pour EAD, changer l'espace de noms TEI par celui d'EAD et le préfixe *tei:* par *ead:*.
- Atributs pour la transformation vers HTML ou LaTeX :
 - *@xmlns:xsl* (adresse de l'espace de noms XSLT)
 - *@xpath-default-namespace* (adresse de l'espace de noms TEI)
 - *@version* (version de XSL utilisée)

XSLT/ Instruction de 1^{er} niveau : <xsl:output>

- Instruction de premier niveau;
- « contrôle les caractéristiques du document de sortie ».

<xsl:output

method="xml | html | text"

indent="yes | no"

omit-xml-declaration="yes | no"

encoding="UTF-8"

/>

XSLT/ Écriture d'une règle : `<xsl:template>` (1/2)

- une règle peut être écrite de plusieurs manières;
- `<xsl:template>` → définir une règle;
- Possède un *`@match`* avec pour valeur un chemin XPath qui donne l'emplacement du nœud sur lequel sera appliquée la règle.

XSLT/ Écriture d'une règle : `<xsl:template>` (2/2)

- Effet d'une règle vide → partie de l'arbre sélectionnée n'est pas copiée dans la sortie.

```
<xsl:template match="/">  
</xsl:template>
```

- Effet de `<xsl:copy-of/>` → copie à l'identique la balise matchée par le `@select` et ses nœuds enfants.

```
<xsl:template match="/">  
    <xsl:copy-of select="."/>  
</xsl:template>
```

XSLT : première approche. Notions essentielles

- Transformation et feuille de style XSL;
- Manipulation de XSL avec Oxygen;
- Élément racine : `<xsl:stylesheet>` et ses attributs (en-tête XSL);
- Instructions de premier niveau :
 - `<xsl:output>`;
 - `<xsl:template>` et son *@match*.

3. XSLT. Règles basiques

XSLT/ `<xsl:template>`

- Un `<xsl:template>` contient une ou plusieurs règles;
- L'*@match* contient le chemin XPath vers la balise qui sera le point de départ de la ou des règles, qui peut s'appliquer à :
 - La balise désignée dans le *@match*;
 - Les enfants ou descendants de cette balise;
 - N'importe quelle balise de l'arbre *via* des axes (*ancestor*, *preceding*, etc.).
- La partie de l'arbre sur laquelle la règle s'applique est en général indiquée dans le *@select* de l'instruction XSL (*cf. infra*).

XSLT/ Contenu d'un `<xsl:template>`

- `<xsl:template>` peut contenir :
 - Des balises XML ou HTML (avec leurs @tt);
 - Du texte (commandes \LaTeX);
 - Des instructions ou variables/param XSL : `<xsl:.../>`;

Contenu d'un `<xsl:template>` : fichier d'exemple

- Ouvrir Oxygen.
- Désactiver le paramètre par défaut :
Preferences > XML > XML Parser > RELAX NG,
décocher *Add default attributes*
- Ouvrir *class2_teiAll_oxygen.xml*.
- Ouvrir une feuille de style XSLT vide,
compléter l'en-tête XSLT et ajouter
l'instruction `<xsl:output/>` (cf. fiche
xsl_header).

- Des balises peuvent être écrites directement dans un `<xsl:template>` :

```
<xsl:template match="//body">
  <body>
    <div>
      <head type="chap">
        <xsl:value-of select="./p[1]"/>
      </head>
      <xsl:copy-of select="./p[2]"/>
    </div>
  </body>
</xsl:template>
```

- Les balises XML ou HTML peuvent être écrites dans des `<xsl:element name="tag">` et les attributs dans des `<xsl:attribute name="@tt">`:

```
<xsl:template match="//body">
  <xsl:element name="div">
    <xsl:element name="head">
      <xsl:attribute name="type">chap</xsl:attribute>
      <xsl:value-of select="./p[1]"/>
    </xsl:element>
    <xsl:copy-of select="./p[2]"/>
  </xsl:element>
</xsl:template>
```

XSLT/ Contenu d'un `<xsl:template>` : du texte (1/2)

- Du texte peut être mis dans une règle : pour remplacer le contenu d'une balise, pour écrire le préambule d'un doc \LaTeX .
- (Dans le fichier d'exemple : remplacer tout.)

```
<xsl:template match="/">
    \documentclass[a4paper]{book}
    \usepackage[utf8]{inputenc}
    \usepackage[french]{babel}
    \usepackage{fontspec}
    \begin{document}
        <xsl:apply-templates/>
    \end{document}
</xsl:template>
```

XSLT/ Contenu d'un `<xsl:template>` : du texte (2/2)

- Du texte peut aussi être mis dans un `<xsl:text>text</xsl:text>`:

```
<xsl:template match="/TEI/text/body">
  <xsl:element name="body">
    <xsl:element name="head">
      <xsl:attribute name="type">chap</xsl:attribute>
      <xsl:value-of select="./p[1]"/>
    </xsl:element>
    <xsl:element name="p">
      <xsl:text>nouveau texte</xsl:text>
    </xsl:element>
  </xsl:element>
</xsl:template>
```

Règles basiques : notions essentielles

- Les balises XML et HTML peuvent être écrites en clair dans un `<xsl:template>`;
- Les balises XML et HTML peuvent aussi être écrites dans un `<xsl:element name="tag">` et un attribut dans un `<xsl:attribute name="@att">`.
- Le texte (commandes \LaTeX) peut être mis soit en clair soit dans un `<xsl:text>`.

4. Les quatre principales instructions XSL

XSLT/ Les quatre principales instructions XSL

- `<xsl:copy/>`
- `<xsl:copy-of/>`
- `<xsl:value-of/>`
- `<xsl:apply-templates/>`

XSLT/ Principales instructions (1/4) : `<xsl:copy>`

- Copie de la balise matchée par le `@match`, sans les *namespaces*, attributs, texte, etc.
- `<xsl:copy/>` peut contenir d'autres règles, du texte, etc.
- Intérêt → copier un élément pour appliquer des règles à ses enfants.

```
<xsl:template match="/TEI/text/body">  
    <xsl:copy>...</xsl:copy>  
</xsl:template>
```

XSLT/ Principales instructions (2/4) : `<xsl:copy-of>`

- Copie à l'identique de la balise matchée par le `@select` et de ses nœuds enfants (balises, attributs, textes).
- Élément vide (pas de règles internes).
- Impossible de modifier les éléments copiés!
- Intérêt → reproduire rapidement une partie de l'arbre que l'on ne veut pas modifier.

```
<xsl:template match="/TEI/text/body">  
  <xsl:copy-of select="."/>  
</xsl:template>
```

XSLT/ Principales instructions (3/4) : `<xsl:value-of>`

- Renvoie uniquement la valeur textuelle de la balise matchée par le `@select`.
- Élément vide (pas de règles internes).
- Intérêt → copier du texte sans les balises (utile pour \LaTeX).

```
<xsl:template match="/TEI/text/body">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- Ouvrir les fichiers
class2_st_julien_hospitalier.xml
et *att_apply_templates_class2.xsl*.
- `<xsl:apply-templates/>` est une instruction réursive : le processeur va examiner les nœuds enfants de la balise matchée par le *@match* dans l'ordre et appliquer les règles qui leur sont associées.

XSLT/ L'élément `<xsl:apply-templates/>` (1/4)

- Sans elle, le processeur s'arrête à l'emplacement désigné par le *@match* du `<xsl:template/>` et ne traite pas les éléments enfants.
- Exemple : la balise `<TEI/>`.

```
<xsl:template match="/">
    <TEI>
        <xsl:apply-templates/>
    </TEI>
</xsl:template>
```

XSLT/ L'élément `<xsl:apply-templates/>` : `@select` (2/4)

- *@select* permet d'appliquer les règles uniquement au nœud sélectionné.
- Ex. → inverser `<teiHeader/>` et `<text/>` (cf. fichier) :

```
<xsl:template match="/">
  <TEI>
    <xsl:apply-templates select="//text"/>
    <xsl:apply-templates select="//teiHeader"/>
  </TEI>
</xsl:template>
```

XSLT/ L'élément `<xsl:apply-templates/>` : @mode (3/4)

- *@mode* permet d'appliquer des règles différentes à un même élément XML en fonction de son emplacement ou de son contenu (= son « mode ») *sans avoir à mettre ces règles dans un seul <xsl:template/>*.
- Le *@mode* doit être présent et dans le `<xsl:apply-templates/>` et dans le `<xsl:template/>` avec la même valeur (= le nom du mode).

XSLT/ L'élément `<xsl:apply-templates/>` : *@mode* (4/4)

- Exemple d'utilisation de *@mode* : créer une table des matières (cf. fichier).

```
<xsl:template match="//body">
  <xsl:copy>
    <xsl:apply-templates />
    <div>
      <head>Table des matières</head>
      <xsl:apply-templates mode="toc"/>
    </div>
  </xsl:copy>
</xsl:template>
```

XSLT/ Une autre instruction : `<xsl:number/>` (1/2)

- `<xsl:number/>` « compte des éléments de façon séquentielle ».
- Attributs :
 - `@count` → définit les éléments de l'input qui seront numérotés dans l'output;
 - `@level="single/multiple/any"` → niveaux de l'arbre pris en compte pour le comptage;
 - `@format="1/01/A/a/I/i"` → format des numéros.

XSLT/ Une autre instruction : <xsl:number/> (2/2)

```
<xsl:template match="//body//p">
  <xsl:copy>
    <xsl:attribute name="n">
      <xsl:number
        count="//div[@n]/p"
        level="any"
        format="1"/>
    </xsl:attribute>
    <xsl:value-of select="."/>
  </xsl:copy>
</xsl:template>
```

XSLT/ Ordre d'application des règles

- XSLT commence par chercher la règle à appliquer au nœud racine;
- Cette règle fait appel à d'autres avec *apply-templates* (avec ou sans @tt), elles sont appliquées dans l'ordre;
- S'il n'y a pas de règle, il passe au suivant.
- Donc : l'ordre d'écriture des règles n'a aucune importance;
- **Attention → restez lisible + commentez votre code!**

5. XPath/ Les fonctions

- Les fonctions peuvent être utilisées dans des expressions XPath ou dans des prédicats :
 - Expression \rightarrow *count*(./p)
 - Prédicat ([]) \rightarrow ./[*count*(p)]
- Cf. fiche : fichier xpath-nodes-axis-functions.

6. XSLT. Variables et paramètres

XSLT/ Différence entre **variable** et **param**

- Rappel → variable : nom (unique) + valeur (statique ou dynamique).
- Dans XSLT, portée globale (premier niveau) ou locale (intégrée à un *xsl:template*);
- *<xsl:variable/> : valeur fixe (NB : la valeur courante peut changer au sein d'une itération);*
- *<xsl:param/> : valeur fixe ou dynamique, peut être pris en argument par le processeur (ex : saxon dans lxml).*

XSLT/ <xsl:variable/> (1/2)

- Attributs : *@name* (req.) et *@select* (opt.);
- Deux manières de définir la valeur de <xsl:variable/> :
 - Par la valeur de l'*@select* (expression XPath);
 - Par le contenu de <xsl:variable/> (texte ou règle).

```
<xsl:variable name="x" select="XPath"/>
```

```
<xsl:variable name="y">[règles]</xsl:variable>
```

XSLT/ <xsl:variable/> (2/2)

- Pour appeler une variable : \$ + @name dans le @select d'un <xsl:value-of/> (uniquement le texte) ou d'un <xsl:copy-of/> (texte + balises).

```
<xsl:template math="/">  
  <xsl:value-of select="$x"/>  
  <xsl:value-of select="$y"/>  
</xsl:template>
```

XSLT/ `<xsl:param/>`

- Fonctionne de la même manière que `<xsl:variable/>`.

7. XSLT. Conditions, boucles et tri

XSLT/ Conditions

- Deux moyens d'exprimer une condition en XSLT : `<xsl:if/>` et `<xsl:choose/>`;
- Attention : les conditions XSLT ne fonctionnent pas de la même manière que celles d'autres langages de programmation, comme Python où l'on trouve le triplet *if*, *elif* et *else*.

XSLT/ `<xsl:if/>` (1/2)

- `<xsl:if/>` possède un *@test* obligatoire;
- Contenu appliqué si le *@test* est validé.
- Pas de *elif* : utiliser plusieurs `<xsl:if/>`;
- Pas de *else* : utiliser `<xsl:choose/>`.

XSLT/ <xsl:if/> (2/2)

Exemple : dans *St Julien*, ajouter un *@n* aux *<p/>* si ces derniers sont dans la *<div/>* n° 2 :

```
<xsl:template match="//div[@n]/p">
  <p>
    <xsl:if test="parent::div/@n = 2">
      <xsl:attribute name="n">
        ...
      </xsl:attribute>
    </xsl:if>
    <xsl:value-of select="."/>
  </p>
</xsl:template>
```

XSLT/ `<xsl:choose/>`

- `<xsl:choose/>` a pour enfant un ou des `<xsl:when/>` (req.) et un seul `<xsl:otherwise/>` (opt.).
- `<xsl:when/>` a un *@test* obligatoire (même fonctionnement que `<xsl:if/>`);
- si aucun `<xsl:when/>` n'est réalisé, alors `<xsl:otherwise/>` est activé

XSLT/ Boucle simple : `<xsl:for-each/>` (1/2)

- `<xsl:for-each/>` a un *@select*;
- Itération sur chaque nœud désigné par le *@select*, la règle contenue dans le `<xsl:for-each/>` est appliquée;
- Le point de départ des chemins Xpath au sein d'un `<xsl:for-each/>` est la balise sélectionnée par le *@select*;
- Peut intégrer d'autres instructions XSL comme `<xsl:if/>` et `<xsl:choose/>`.

XSLT/ Boucle simple : <xsl:for-each/> (2/2)

- Exemple :

```
<xsl:for-each select="//persName">
  <persName>
    <xsl:value-of select="./forename"/>,
    <xsl:value-of select="./surname"/>.
  </persName>
</xs:for-each>
```

XSLT/ Tri : `<xsl:sort/>` (1/2)

- S'utilise comme premier enfant de `<xsl:for-each/>` ou `<xsl:for-each-group/>`;
- Change l'ordre d'origine des nœuds sélectionnés en un autre ordre (alphabétique, numérique, etc.);

- Exemple :

```
<xsl:for-each select="//date[@when]">
  <xsl:sort select="./when"
            data-type="number"/>
  <xsl:copy-of select="."/>
</xs:for-each>
```

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (1/3)

- Rassemble les nœuds (*@select*) en groupe selon un critère donné (*@group-by*) et leur applique les règles définies à l'intérieur de `<xsl:for-each-group/>`.
 1. Itère sur les nœuds désignés dans le *@select*;
 2. Regroupe ces nœuds selon le critère défini dans le *@group-by*;
 3. Applique les règles indiquées dans le contenu de la balise.

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (2/3)

- Plusieurs `<xsl:for-each-group/>` sont souvent utilisés ensemble (boucles dans une boucle);
- Ils peuvent contenir des `<xsl:for-each/>`, des conditions, etc...
- Attention : c'est une nouveauté de XSLT 2.0.

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (3/3)

- *current-group()* retourne la liste des nœuds du groupe de l'itération en cours d'exécution;
 - à utiliser dans le *@select* d'un sous-`<xsl:for-each-group/>` ou d'un `<value-of/>`.
- *current-grouping-key()* retourne la clef utilisée pour l'itération en cours d'exécution.
 - à utiliser dans le *@select* d'un `<value-of/>`.

XSLT/ Conditions et boucles. Notions essentielles

- Conditions XSL :
 1. Sans alternative : `<xsl:if/> (@test);`
 2. Avec alternative : `<xsl:choose/>`, contient des `<xsl:when> (@test)` et un `(<xsl:otherwise/>)`.
- Boucles XSL :
 1. Simple : `<xsl:for-each/> (@select);`
 2. Sur des groupes : `<xsl:for-each-group/> (@select, @group-by).`
- Tri XSL : `<xsl:sort/> (@select).`

8. Appel d'une template :
`<xsl:call-template/>`

XSLT/ Instruction `call-template` (1/2)

- `<xsl:call-template name="#"/>`;
- Permet d'appeler autant de fois qu'on le veut une template dans un `<xsl:template/>`
- Cas pratiques :
 - Un *call-template* par *result-document*;
 - Peut permettre de stocker des parties du document de sortie (header, footer) au même titre que les variables.

XSLT/ Instruction call-template (2/2). Exemple.

```
<xsl:template match="/">
  <xsl:call-template name="name1"/>
</xsl:template>
```

```
<xsl:template name="name1">
  <xsl:result-document href="#">
    <html>
      ...
    </html>
  </xsl:result-document>
</xsl:template>
```

9. XSLT. Enregistrement du document de sortie

- `<xsl:result-document/>` permet de générer plusieurs documents de sortie;
- Nouveauté de XSLT 2.0;
- S'utilise comme enfant d'un `<xsl:template/>` ou d'un `<xsl:call-template/>`.

Trois attributs :

- *`@href`* permet d'indiquer l'URI du document à générer : absolue (depuis la racine) ou relative (dans ce cas, le processeur part de l'emplacement du fichier d'entrée);
- *`@method`* permet de spécifier le format du code de sortie;
- *`@indent`* permet d'indiquer si le résultat sera indenté;

XSLT/ <xsl:result-document/>, exemple

```
<xsl:template match="/">
  <xsl:result-document href="st_julien1.xml"
    method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
  <xsl:result-document href="st_julien2.xml"
    method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
  <xsl:result-document href="index.xml"
    method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
</xsl:template>
```

XSLT/ `<xsl:result-document/>` et variables

- Inconvénient de créer X docs de sortie → répéter X fois des éléments récurrents (`<teiHeader/>`, header HTML, etc.);
- Stocker ces éléments dans des variables permet de ne les modifier qu'une seule fois.

XSLT/ <xsl:result-document/> et URI

- Bonne pratique de programmation → déclarer les URI des docs de sortie dans des variables au début de la feuille de style.

```
<xsl:variable name="chap1">
    <xsl:value-of select="
        concat('st_julien_chap1', '.xml')"/>
</xsl:variable>
<xsl:template match="/">
    <xsl:result-document href="$chap1"
        method="xml" indent="yes">
        <TEI><!-- règles --></TEI>
    </xsl:result-document>
</xsl:template>
```

XSLT/ `<xsl:result-document/>`. Notions essentielles

- `<xsl:result-document/>` permet de générer un ou plusieurs documents de sortie;
- L'URI est indiquée dans un *`@href`*;
- Les variables sont utiles pour :
 - Déclarer les URI des documents de sortie au début de la feuille de style;
 - Contenir les éléments récurrents des documents de sortie (header, footer, etc).

10. XSLT. Transformation vers HTML

- Dans l'en-tête XSL : les *@xmlns:tei*, *@xmlns* et *@exclude-result-prefixes* ne sont plus nécessaires;
- Dans *<xsl:method/>* : ajouter *@method='html'*, *@omit-xml-declaration* n'est plus nécessaire.

XSLT/ Méthode HTML (2/2). Template HTML basique

```
<html lang="fr">
  <head>
    <meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
    <!-- le 1er <meta> est ajouté par XSLT -->
    <title><!-- doc title --></title>
    <meta name="description" content="#"/>
    <meta name="author" content="#"/>
    <!-- <link rel="stylesheet" href="#"/> -->
  </head>
  <body>
    <!-- some text here -->
    <!-- <script src="#"></script> -->
  </body>
</html>
```

11. XSLT. Transformation vers \LaTeX

XSLT/ Méthode texte (1/2)

- Même en-tête XSL que pour la transformation vers HTML.
- Transformation directe de XML à \LaTeX non-prise en charge par XSLT → **utiliser la méthode `text` dans le `<xsl:output/>`.**
- Conséquence → les commandes \LaTeX ne sont pas reconnues comme telles et traitées comme du texte.

XSLT/ Méthode texte (2/2). Template \LaTeX basique

```
\documentclass[]{book}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\title{}
\author{}
\date{}

\begin{document}
\maketitle
% votre texte
\tableofcontents
\addcontentsline{toc}{chapter}{Table des matières}
\end{document}
```