

XML : Ressources et Manipulation

Lydia-Mai Ho-Dac

Ressources linguistiques pour le TAL, Master LITL, UT2J



Table des matières

1	XML c'est quoi ?	2
1.1	Exemples de XML : les flux rss (<i>Really Simple Syndication</i>)	2
1.2	Exemples de XML : les sitemap	3
1.3	Syntaxe de base d'un document XML	4
1.4	Outillage XML	6
2	Définir la grammaire d'un document XML : DTD ou XSD	9
2.1	Syntaxe de base d'une DTD	9
2.2	Norme TEI-P5	11
2.2.1	Encodage des métadonnées principales	12
2.2.2	Encodage de la structure de document	12
3	XPath : navigation dans un document XML	15
3.1	Syntaxe des expressions XPATH	15
4	Transformations de document XML : XSLT	19
4.1	XMLNS : la notion de " <i>namespaces</i> " (<i>espaces de nommage</i>)	19
4.2	Syntaxe d'une feuille de transformation XSLT	19
4.3	Méthodes de transformation XSLT	22
4.3.1	Utilisation d'un éditeur XML	23
4.3.2	Utilisation d'un script de transformation	24
5	Liens	25

1 XML c'est quoi ?

XML (*eXtensible Markup Language* – langage de balisage extensible) appartient à la famille des **langages de balisage** (*Markup Languages*) comme HTML (*HyperText Markup Language*). XML et HTML sont des dérivés du langage SGML (*Standard Generalized Markup Language*) correspondant à la norme internationale (norme ISO¹ 8879 : 1989²)

- lisible par l'être humain et traitable par une machine
- permettant de définir des langages de balisage (ce que sont HTML et XML)

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <tagA>
    text
  </tagA>
</root>
```

Tutoriel : <http://www.w3schools.com/xml/>

SGML	1983	Le <i>Standard Generalized Markup Language</i> (Charles Goldfarb, ex-IBM) a pour objectif normaliser la structure des documents qui sont généralement utilisés dans le monde industriel. Le principe de base est de séparer les données de leur description. Le langage SGML est alors conçu pour décrire les données et leur structure via un langage de balisage générique et normalisé et donc partageable par la communauté.
HTML	1991	L' <i>Hyper-Text Markup Language</i> (Tim Berners-Lee) est une application du langage SGML visant à proposer un langage très simple à utiliser, non extensible et rapide à apprendre (à l'inverse de SGML trop difficile à maîtriser sans formation) facilitant la navigation dans le web.
XML	1998	L' <i>eXtensible Markup Language</i> (World Wide Web Consortium) est également dérivé du langage SGML. Il répond au besoin de simplifier SGML pour traiter (dynamiquement ou non) les données partagées sur le web.

Tableau 1 – SGML, HTML, XML

1.1 Exemples de XML : les flux rss (*Really Simple Syndication*)

Le flux RSS sont définis comme une "Syndication" de contenu web i.e. normalisation (selon un modèle commun) du format de diffusion de résumés d'information. L'objectif d'une telle normalisation est la diffusion multi-support (mobile, ordinateur) et l'agrégation de contenus provenant de différentes sources (cf. <http://rssnewsbox.com/fr>).

-
1. Organisation internationale de normalisation
 2. date de baptême

HTML	XML
balises standards (langage prédéfini)	balises à définir (langage extensible)
inventé pour consulter le web : mise en forme l'info et hyperliens	inventé pour structurer et décrire les données afin de les traiter
syntaxe laxiste	structure arborescente et nécessairement bien formée i.e. syntaxe stricte
balises de mise en forme de l'information	description fonctionnelle sans signification a priori
<code><h1></h1></code> ou <code><H1></H1></code>	<code><titre></titre></code> ou <code><head></head></code> ou <code><smurtz></smurtz></code>
confusion entre fond et forme (données et mise en forme)	distinction nette entre données et structure : les balises indiquent sur la fonction des éléments

Tableau 2 – HTML vs. XML

Exercice 1 : Structure normalisée des flux RSS.

Parcourez les flux suivants en affichant le code source de la page et dégagez les éléments (balises) communs qui apparaissent ^a.

- <http://www.lemonde.fr/rss/> (cliquer sur un des boutons)
- <http://www.spiegel.de/schlagzeilen/index.rss>
- <http://www.infoclimat.fr/photolive/rss.php>
- <http://midi-pyrenees.france3.fr/actu/rss>
- <https://linguistlist.org/issues/rss/topics.cfm> (cliquer sur un des boutons)

^a. Exemple sur la page Wikipedia : RSS, consulté le 27 fev. 2013 ([lien](#))

1.2 Exemples de XML : les sitemap

Extrait du site <http://www.sitemaps.org/fr/>

"Sitemaps permet aux webmasters d'indiquer facilement aux moteurs de recherche les pages de leurs sites à explorer. Dans sa forme la plus simple, un plan Sitemap est un fichier XML qui répertorie les URL d'un site ainsi que des métadonnées complémentaires sur chaque URL (date de dernière modification, fréquence de révision et importance relative par rapport aux autres URL du site), de façon à favoriser une exploration plus intelligente du site par les moteurs de recherche."

Pour afficher dans un navigateur web le plan *Sitemap* d'un site web, il suffit d'ajouter `sitemp.xml` à la fin de l'url **racine** d'un site.

Exercice 2 : Plan *sitemap* des sites <http://www.sitemaps.org/sitemap.xml>.

Essayez d'afficher le plan *Sitemap* de la page <http://www.sitemaps.org/fr/>. Quel est le problème ?

Exercice 3 : Structure normalisée d'un plan Sitemap.

Observer les plans *Sitemaps* de différents sites et dégager les éléments (balises) communs qui apparaissent. Exemples de site à visiter :

- <http://www.franceinter.fr/>
- <http://bling.hypotheses.org/>
- <http://payetonsport.tumblr.com>

Observer ce qui est indiqué sur la page : <https://www.sitemaps.org/fr/protocol.html>

1.3 Syntaxe de base d'un document XML

Un document XML doit être "bien formé" c'est à dire respectant des règles de syntaxe fondamentales.

- Le document XML doit contenir une déclaration (voir infra) qui doit obligatoirement être placée en première ligne du document (sans même un espace avant).

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

- Les balises suivent la syntaxe suivante :

```
<nomDeBalise attribut="valeur">texte</nomDeBalise>
```

- Un document XML ne doit avoir qu'un et un seul élément appelé "élément racine", qui doit contenir tous les autres éléments

```
<elementRacine>
  <nomDeBalise attribut="valeur">
    texte
  <baliseAutoFermante/>
</nomDeBalise>
</elementRacine>
```

- Le nom des balises est uniquement formé de lettres non accentuées [a-z|A-Z], de chiffres [0-9] et/ou des symboles "- _ ."
- Les éléments peuvent s'emboîter :

```
<a>
  <b>

  </b>
</a>
```

mais pas se chevaucher :

```
<a>
  <b>
</a>
  </b>
```

- L'ordre des éléments a de l'importance (car il implique des relations d'emboîtements et donc de hiérarchie – enfant/parent).
- L'ordre des attributs n'en a pas :

```
<a attr1="1345" attr2="fer">
```

équivalent complètement à

```
\lstinline|<a attr2="fer" attr1="1345">|.
```

Remarques

- `<a>` est équivalent à `<a/>` ("balise auto-fermante")
- On peut écrire des commentaires en les encadrant de balises `<!-- comment -->` (comme en HTML)
- Certains balises et nom d'attributs peuvent contenir deux point (:). Il s'agit alors d'éléments propres à un espace de nommage (voir 4.1).

Balise (ou tag ou label)

Marque de début et fin permettant de repérer un élément textuel

Forme : `<balise>` de début, `</balise>` de fin

Élément de données

Texte encadré par une balise de début et une de fin. Les éléments de données peuvent être imbriqués (donc un élément peut contenir des balises)

```
<producteur>
  <adresse>
    <rue>A. Briand</rue>
    <ville>Dijon</ville>
  </adresse>
</producteur>
```

Attribut

nom="valeur" qualifiant une balise

```
<producteur no="160017" region="Bourgogne">
  <adresse type="peronnelle">
    <rue>A. Briand</rue>
    <ville>Dijon</ville>
  </adresse>
  <adresse type="professionnelle">
    <rue>R. de la Paix</rue>
    <ville>Paris</ville>
  </adresse>
</producteur>
```

Entités

Certains caractères ne peuvent pas être utilisés dans le texte des éléments.

Ex : < et &

Leur mention réalise alors un **appel d'entités** : `&nom_de_l_entité` ; sorte de variable.

Pour utiliser ces caractères sans faire d'appel d'entités, il faut utiliser les entités prédéfinies qui leur correspondent (les mêmes qu'en HTML) :

```
&lt; ; <
&amp; ; &
&gt; ; >
&quot; ; "
&apos; ; '
```

On peut aussi utiliser le segment `<![CDATA[j'&cris_ce_que <je_veux>]]>`

Déclarations et Instructions de traitements

Déclarations : lignes qui déclarent un certain nombre de variables propres au document.

format `<? ... ?>`

— Différentes des balises qui nécessite une fermeture, il n'y a pas de `</? ... ?>`.

exemple la première ligne de tout document XML est une déclaration qui indique que le document est du XML et informe du numéro de version du XML (attribut "version"), de l'encodage des caractères du document (attribut "encoding").

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

L'information `standalone="yes"` signifie qu'aucune DTD externe n'est associée au document. Si une DTD est associée au document, il faut alors indiquer `standalone="no"` et spécifier sa localisation dans une ligne particulière.

Instructions de traitements : lignes qui permettent de déclarer un certain nombre d'instructions.

format `<! ... >`

— Différentes des balises qui nécessite une fermeture, il n'y a pas de `</! ... >`.

exemple ligne qui indique que le document traiter selon une "grammaire" spécifique (une DTD – Document Type Description).

```
<!DOCTYPE elementRacine SYSTEM "fichier.dtd">
```

DOCTYPE et SYSTEM sont des mots réservés, elementRacine précise le nom de l'élément racine du document et fichier.dtd est le chemin d'accès au fichier DTD associé. Ce chemin désigne un fichier local à la machine ou une URL. La DTD peut également être accessible via internet. Dans ce cas, le mot-clé PUBLIC remplace SYSTEM, ce qui indique que la DTD est situé sur un serveur :

```
<!DOCTYPE elementRacine PUBLIC "http://.../fichier.dtd">
```

Exercice 4 : Mon premier XML - 1.

Créer librement un document xml. Tout d'abord dessinez un arbre représentant la structure de votre document, puis construisez un extrait de document conforme à cette structure.

1.4 Outillage XML

Les fichiers XML peuvent se lire via une grande variété d'outils puisqu'ils sont finalement de simples fichiers de texte brut agrémentés de balises (également écrites en texte brut).

- Tout afficheur de texte (ex : un navigateur Web) peut permettre d'afficher des fichiers XML s'ils sont bien formés. Si un document XML est mal formé, le navigateur web vous indiquera les problèmes rencontrés et n'affichera pas le document.
- Tout éditeur de texte peut permettre d'éditer des fichiers XML (mais attention aux éditeurs de type Word et LibreOffice qui peuvent les convertir par défaut dans un format propriétaire). Contrairement à un navigateur, l'éditeur de texte ne vérifie pas la bonne formation d'un document XML.
- Des outils dédiés permettent de manipuler des fichiers XML c'est-à-dire de vérifier et analyser la syntaxe du fichier (*parser*), de sélectionner certaines données et de les extraire, de créer un nouveau document (XML ou HTML ou autre) en transformant la structure

des données ou en ne sélectionnant que certaines données, etc. Certains sont de simples boîtes à outils comprenant des fonctions utilisables en ligne de commande (nous verrons **xmllint**), d'autres proposent une interface conviviale (ex : **OxYgen**). De plus, il existe une grande quantité de modules associés aux différents langage de programmation (les M1 LITL pourront voir certains modules Python dans le cours d'*informatique pour le TAL 2*)

Outillage XML sous NotePad++ – OS Windows : télécharger et installer depuis le site officiel : <https://notepad-plus-plus.org/fr/>. **Lors de l'installation cocher l'option permettant de charger des plugins.**

Installer les modules complémentaires dédié au XML (Compléments → plugin Manager → Show Plugin Manager) :

- *XML Tools* pour gérer, vérifier et explorer (XPath) des documents XML
- *Preview HTML* pour visualiser des documents HTML

Installation : Vérifier la bonne formation d'un document XML avec NotePad++.

```
Plugins > XML Tools > Evaluate XPath expression
```

Installation : Indenter un fichier XML avec NotePad++.

```
Plugins > XML Tools > Pretty Print
```

Outillage XML sous Eclipse : *Web Tools Platform* disponible sous Eclipse : https://wiki.eclipse.org/WTP_FAQ#How_do_I_install_WTP.3F.

Installer les modules :

- *Eclipse XML Editors and Tools*
- *Eclipse XSL Editors and Tools*
- *Web Page Editor*

Dépôt utilisé pour ce cours (Eclipse version luna) : <http://download.eclipse.org/webtools/repository/luna/>

XMLLINT : Outil XML en ligne de commande <http://xmlsoft.org/xmllint.html>.

Installation via le paquet libxml2-utils : `sudo apt-get install libxml2-utils`.

Installation : Parser.

Parser (vérifier la bonne formation) d'un document XML en affichant ou pas son contenu (l'option "-noout" permet de ne pas afficher en sortie le document parsé)

```
xmllint --noout fichier.xml
```

Lancer un navigateur XML dans le terminal (le fichier devient alors une arborescence à la manière d'un dossier) et ensuite valider

```
xmllint --shell fichier.xml  
> validate
```

Installation : Indenter un fichier.

Vous pouvez également utiliser la commande `--format` pour indenter un document. Testez avec le flux de la revue Wired ([lien](#)) sortie STDOUT

```
xmllint fichier.xml --format
```

sortie dans un nouveau fichier

```
xmllint fichier.xml --format --output fichier_indented.xml
```

Remarque : XMLELINT ne gère pas les fichiers XML contenant des *namespaces* (voir section 4.1)

Exercice 5 : Mon premier XML - 2.

1. Vérifier avec NotePad++ ou XMLELINT la bonne formation de votre document XML créé lors de l'exercice 4
2. Indenter ce fichier avec NotePad++ ou XMLELINT

Installation : Help.

Pour voir toutes les commandes possibles ^a :

```
xmllint --shell fichier.xml  
> help  
> exit
```

ou

```
xmllint --help
```

^a. Une liste avec mise en forme plus lisible est également disponible [ici](#) (consulté le 17 fev. 2014). Sur son blog, Nicolas Thouvenin avait posté en 2005 un résumé en français des principales fonctions ([lien](#), consulté le 27 fev. 2013)

Exercice 6 : Vérifier la bonne formation de fichiers XML.

Parser et corriger les fichiers AgoraVox_XXX.xml déposés sur IRIS

2 Définir la grammaire d'un document XML : DTD ou XSD

Un document XML peut être associé à différents éléments qui vont définir la grammaire du document (l'arborescence, les éléments et attributs nécessaires et facultatifs ainsi que le type de données accepté) : soit une DTD (Document Type Definition) soit un XSD (XML Schema Definition). Une DTD tout comme un schéma décrit les balises autorisées (et leur structure). Un schéma XSD est un fichier au format XML alors qu'une DTD est un fichier au format texte. Tous deux permettent de définir :

- un vocabulaire (les balises éléments et leurs attributs)
- une grammaire (les règles d'imbrication i.e. l'arborescence)

Un document XML conforme aux définitions données dans une DTD ou un schéma est appelé document **valide**. L'association entre un fichier XML et une DTD se fait via une information de traitement en début de fichier, après la déclaration XML :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE elementRacine SYSTEM "fichier.dtd">
<elementRacine>
  <nomDeBalise attribut="valeur">
    texte
    <baliseAutoFermante/>
  </nomDeBalise>
</elementRacine>
```

DOCTYPE et SYSTEM sont des mots réservés, `elementRacine` précise le nom de l'élément racine du document et `fichier_dtd` est le chemin d'accès au fichier DTD associé. Ce chemin désigne un fichier local à la machine ou une URL. La DTD peut également être accessible via internet. Dans ce cas, le mot-clé `SYSTEM` est remplacé par `PUBLIC` qui spécifie la localisation de la DTD via un URL :

```
<!DOCTYPE nom_element_racine PUBLIC nom_dtd_quelconque "http://fichier.dtd">
```

Exercice 7 : Exemples de ressources XML associées à une DTD.

- ressources sur le site [REDAC](#) : [lexiques](#) et [corpus](#)
- lexiques Glawi et wiktionaryX
- ressources sur le site du [CNRTL](#) : [lexiques](#) et [corpus](#)
- lexique Morphalou
- corpus manuscrit

2.1 Syntaxe de base d'une DTD

Une DTD est un fichier au format texte enregistré avec l'extension `.dtd`.

Définition des éléments

La définition d'un élément ELEMENT s'écrit :

```
<!ELEMENT balise (contenu)>
```

ce qui permet de définir ce qui peut être contenu dans un élément de nom "balise". Le paramètre "contenu" représente soit un type de donnée prédéfini, soit un ensemble d'éléments-enfants (récursivité)

Types de données prédéfinis :

- **ANY** : l'élément peut contenir tout type de donnée ; ex :

```
<!ELEMENT segment ANY>
```

- **EMPTY** : l'élément ne contient pas de données spécifiques ; ex :

```
<!ELEMENT alinea EMPTY>
```

- **#PCDATA** : l'élément **doit** contenir une chaîne de caractère ; contrairement à ANY et EMPTY, il s'écrit entre parenthèses ex :

```
<!ELEMENT phrase (#PCDATA)>
```

Définition des éléments-enfants constitutifs de **ELEMENT** :

- Si les éléments-enfants doivent apparaître selon un ordre fixé : séparation des éléments-enfants par une **virgule** (,) ; ex :

```
<!ELEMENT article (titre, paragraphe, liste, figure, tableau)>
```

- Si il n'y a pas d'ordre fixé pour l'apparition des éléments-enfants : séparation des éléments-enfants par un **pipe** (|) ; ex :

```
<!ELEMENT listItem (paragraphe | liste)>
```

Chaque (groupe d') élément(s)-enfant(s) peut-être suivi d'un opérateur indiquant son nombre d'occurrence. Ne pas indiquer cette information signifie que l'élément est obligatoire et ne doit apparaître qu'une seule fois. Les opérateurs sont les suivants :

*	zéro ou plusieurs fois
+	une ou plusieurs fois <code><!ELEMENT liste listItem+></code>
?	une fois au plus

L'élément peut également être défini par un mélange de type de données et d'éléments-enfants ; ex :

```
<!ELEMENT paragraphe (#PCDATA | quote | liste)*>
```

Définition des attributs

Les attributs éventuels d'un élément s'indiquent en précisant le nom de l'élément dont ils sont attributs, leur type et leur statut : obligatoire (**#REQUIRED**), optionnel (**#IMPLIED**) ou à valeur prédéfinie et fixe (**#FIXED**) ; ex :

```
<!ATTLIST nom_element
  nom_attribut1 TYPE #REQUIRED
  nom_attribut2 TYPE #IMPLIED
  nom_attribut3 TYPE #FIXED "xxx"
>
```

Les principaux types d'attributs sont :

- **CDATA** : chaîne de caractères
- (**val1** | **val2** | ... | **valn**)
- **ID** : une valeur donnée unique

Définition d'entités

Comme indiqué précédemment (1.3), certains caractères comme le & doivent se réaliser sous la forme d'un **appel d'entités** (ex : &);).

Une DTD peut définir des entités propres en plus des entités prédéfinies ; ex :

```
<!ENTITY nom_entite "xxx">
```

Ainsi, toute occurrence de &nom_entite; dans un document XML ou dans la DTD elle-même sera remplacé par le texte *xxx*.

Installation : Valider avec XMLLINT (linux).

Selon une DTD (.dtd)

```
xmllint --noout --dtdvalid fichier.dtd fichier.xml
```

ou si la DTD est indiquée en en-tête du document XML :

```
xmllint --noout --valid fichier.xml
```

Selon un schéma (.xsd)

```
xmllint fichier.xml --schema fichier.xsd --noout
```

Installation : Valider avec une application java.

Benoît Valiron propose plusieurs petits scripts sur son site : allez sur <http://www.monoidal.net/inf356/validation.html>, télécharger le script java *validation.jar* et suivez les instructions

Installation : Valider avec une application en ligne.

Allez sur <http://www.xmlvalidation.com/> et suivez les instructions

Installation : Valider avec NotePad++.

Uniquement selon un schéma (.xsd)

```
Plugins > XML Tools > Validate Now
```

2.2 Norme TEI-P5

La norme Text Encoding Initiative a été créée en 1990 (version P1) par concertation entre universitaires et éditeurs dans le but de proposer une norme de codage des textes utilisés en sciences humaines (tous types/langues/époques).

Site du consortium : <http://www.tei-c.org> et les instructions de la TEIP5 <http://www.tei-c.org/Guidelines/P5/>

Norme = nommer tous les différents segments et attributs dont on peut avoir besoin pour caractériser les données présentes dans des textes utilisés en sciences humaines.

Exemples :

Métadonnées : auteur, éditeur, date, traductions, etc. (plusieurs dizaines de champs)

Corps du texte : — En fonction d'une classe de document (prose, vers, théâtre, oral, dictionnaire, terminologie, général et divers)

- Annotation structurelle (chapitre, paragraphe, section, phrase, etc.)
- Annotation de contenu (tableaux, dessins, formules)

Édition : (versions, traductions, notes, renvois, datation, imprécisions, ratures, etc.)

Un document normé selon la TEI-P5 est composé obligatoirement d'une racine `<TEI>` présente deux parties principales :

1. la partie `<teiHeader>` : partie du document contenant toutes les métadonnées i.e. les informations relatives à la description du fichier et aux caractéristiques de l'encodage selon la TEI-P5 du document.
2. la partie `<text>` correspondant au corps du texte qui est lui-même subdivisé en 3 parties principales : la partie `<body>` correspondant au corps de texte et deux parties facultatives : une partie `<front>` correspondant aux éléments textuels précédant le corps de texte en lui-même (ex : la page de titre, les préfaces, la tables des matières, etc.) et une partie `<back>` correspondant aux éléments textuels suivant le corps de texte en lui-même (ex : la bibliographie, les notes de fin, les annexes, les postface, etc.)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<TEI>
  <teiHeader>
  </teiHeader>
  <text>
    <front>
    </front>
    <body>
    </body>
    <back>
    </back>
  </text>
</TEI>
```

2.2.1 Encodage des métadonnées principales

Description du fichier (`fileDesc`) Cette partie du `header` consiste à décrire le fichier XML : sa source, les différents traitements subit par le document source, et l'institution diffuseur du fichier :

- la source à partir de laquelle a été construit le fichier XML (`sourceDesc`)
- la référence et les traitements subit par le document pour devenir le fichier .xml, ainsi que leurs responsables (`titleStmt`)
- le cadre institutionnel par lequel le fichier .xml est diffusé (`editionStmt`)

Caractéristiques d'encodage (`encodingDesc`) Cette partie du `header` consiste à décrire les caractéristiques définissant l'encodage du document :

- le projet dans lequel s'intègre cet encodage (`projetDesc`)
- les catégories "méta" dans qui caractérisent le document i.e., genre, discipline (`classDecl`)

Autres informations concernant le profil du document (`profileDesc`) Cette partie permet par exemple d'indiquer la langue du document.

2.2.2 Encodage de la structure de document

Exemple de modèle d'encodage de la structure de document des articles d'AgoraVox :

```
<TEI>
  <teiHeader>
</teiHeader>
```

text = texte de l'article

```
<text>
```

front = zone de texte précédant l'article en soi et contenant diverses informations sur le texte mais apparaissant dans la version publiée

```
<front>
```

titlePage = page de titre (crée automatiquement, correspond grosso-modo au bandeau sur les pages web)

```
<titlePage>
  <p></p>
  <p></p>
  <docAuthor><name></name></docAuthor>
  <docEdition>
    <date></date>
  </docEdition>
  <docTitle>
    <titlePart type="main">
      Eric Raoult dans le texte : vingt ans de perles droitieres
    </titlePart>
  </docTitle>
</titlePage>
</front>
```

body = texte de l'article en soi

```
<body>
```

les figures, extraits audio, tableaux, vidéos et entretiens/interviews font l'objet de balises auto-fermantes qui signalent leur présence mais n'en donnent aucun détail

```
<figure/>
<table/>
<video/>
<audio/>
<div type="entretien"/>
<div type="poeme-chant"/>
```

div type="chapo" = chapeau qui annonce l'article tout en attirant le lecteur

```
<div type="chapo">
```

div type="meta"> qui encadre les informations méta apparaissant dans le body e.g. Londres, de notre correspondant

```
<div type="meta">
  <p>Kisangani</p>
  <byline>De notre envoy e sp ciale</byline>
</div>
```

paragraphe simple

```
<p></p>
</div>
```

div type="level_n" = section de niveau n (généralement niveau 1, rarement de niveau 2, jamais de niveau 3)

```
<div type="level_1">
```

bold et ital = deux mises en forme de caractère possibles

```
<p><bold>xxxxxx</bold>xxxx <ital>xxxx</ital> xxx</p>
</div>
<div type="level_1">
```

head = titre de section

```
<head></head>
<p></p>
```

cit = citation

```
<cit>
```

quote = division minimale dans une citation = paragraphe

```
<quote></quote>
</cit>
```

list type="bulleted" = liste avec puces

```
<list type="bulleted">
```

item = item, élément d'une liste. Ces éléments peuvent contenir des subdivisions sous forme de paragraphes, de citations, de listes, etc.

```
<item></item>
<item><p></p><p></p></item>
<item><cit><quote></quote></cit></item>
</list>
```

list type="ordered" = liste numérotée

```
<list type="ordered">
</list>
```

les signatures de l'auteur ou la mention du photographe sont laissées comme des paragraphes simples dans le corpus du texte

```
<p>photo de X</p>
</div>
</body>
</text>
</TEI>
```

A tester *Roma*, interface de génération de DTD dédiée : <http://tei.oucs.ox.ac.uk/Roma/>

Exercice 8 : norme TEIP5.

Adaptez votre premier XML à la norme TEIP5 (selon modalités données en cours)

3 XPath : navigation dans un document XML

XPath est un langage de navigation utilisé pour trouver une information dans un document XML. Il permet de définir les chemins pour parcourir l'arborescence d'un document XML, à la manière des chemins dans l'arborescence du système de fichiers d'un ordinateur (commandes *cd* – *change directory*). En plus de spécifier les éléments recherchés par leur position dans l'arborescence, il permet de spécifier certains critères pour filtrer les éléments recherchés selon leur contenu et leurs attributs.

tutoriel : <http://www.w3schools.com/xpath/>

Installation : Requêtes XPath avec NotePad++.

Plugins > XML Tools > Evaluate XPath expression

Entrer la requête dans l'éditeur et cliquer sur le bouton

Installation : Requêtes XPath avec XMLLINT.

```
xmllint --xpath "expr" fichier.xml
```

Pour plus de rapidité, possibilité de créer un alias

```
alias xq="xmllint --xpath"
```

Ce qui permet de réduire la ligne de commande précédente en

```
xq "expr" fichier.xml
```

RQ : si vous êtes en terminal de navigation xmllint (`xmllint --shell fichier.xml`) il suffit de commander

```
xpath expr
```

(sans les guillemets!)

3.1 Syntaxe des expressions XPATH

reprise du mémento rédigé par Franck Sajous :

3 XPath

XPath désigne la syntaxe utilisée pour adresser des éléments particuliers d'un document XML en fonction de leur structure et/ou de leur contenu.

3.1 Filiation

La sélection relative à la structure utilise une syntaxe similaire aux systèmes de fichiers Unix. Deux caractères spéciaux sont utilisés :

- / permet de séparer les éléments de manière hiérarchique (père/fils). Utilisé en début d'expression, ce caractère représente la racine du document ;
- .. désigne la parent de l'élément courant
- * représente un élément de nom quelconque.

/	racine du document	<pre><r> ... </r></pre>
/r	Racine du document. Le nom de la racine est r	<pre><r> ... </r></pre>
/r/f	élément f, fils de la racine r	<pre><r> <f>...</f> </r></pre>
//e	élément e, où qu'il se trouve	<pre><...> ... <e> ... </e> ... </...></pre>
//e/f	élément fils d'un élément e, où que se trouve l'élément e	<pre><...> ... <e> <f> ... </f> </e> ... </...></pre>
//e/*	tout élément fils d'un élément e	<pre><...> ... <e> <...> ... </...> </e> ... </...></pre>
//e/*	tout descendant d'un quelconque élément e	<pre><...> ... <e> ... <...> ... </...> ... </e> ... </...></pre>
//e/@attr	attribut attr d'un élément e	<pre><...> ... <e truc="..." attr="valeur">...</e> ... </...></pre>
//e[@attr="val"]/..f	tous les éléments f frères de (tout élément e possédant un attribut attr de valeur val)	<pre>... <p> <e truc="..." attr="val">...</e> <f> ... </f> <x> ... </x> <f> ... </f> </p> ...</pre>

3.2 Clauses sur les éléments et les attributs

On peut restreindre la sélection des éléments à l'aide d'expressions placées entre crochets à la suite du nom des éléments :

```
//nom_element[contraintes]
```

On peut combiner plusieurs contraintes avec les opérateurs logiques `and`, `or` et `!` (not). Les contraintes peuvent porter sur la valeur textuelle ou numérique des éléments fils ou sur celles des attributs (dont le nom est précédé par le symbole `@`). La sémantique des opérateurs utilisés est parfois interprétée implicitement en fonction du contexte. Par exemple, l'opérateur `>` (supérieur) fera référence à l'ordre arithmétique si les éléments comparés représentent des nombres, à l'ordre lexicographique en présence de chaînes de caractères.

Un certain nombre de fonctions et opérateurs sont également mis à disposition. Nous pouvons en citer quelques-uns :

contains(string1, string2)	Sélectionne l'élément courant si <i>string1</i> contient la sous-chaîne <i>string2</i> . Cette fonction est sensible à la casse.
starts-with(string1, string2)	Sélectionne l'élément courant si <i>string1</i> commence par <i>string2</i>
string-length(string)	renvoie la longueur de la chaîne <i>string</i>
position()	position de l'élément courant (position par rapport à l'élément père, dans la liste des fils)
text()	Désigne l'ensemble des éléments textes fils de l'élément courant
div	$x \div y$ renvoie le résultat de la division réelle de x par y
mod	$x \bmod y$ renvoie le résultat de la division réelle de x par y
<, >, <=, >=, =, !=	supérieur, inférieur, inférieur ou égal, supérieur ou égal, égal, différent
.....	

La liste exhaustive se trouve sur <http://www.w3.org/TR/xpath>

À propos de text()

Dans le cas du document suivant :

```
<date>
  <jour>9</jour>
  <mois>juin</mois>
  <annee>2004</annee>
</date>
```

L'expression `//date/text()` évaluée dans oXygen provoque l'apparition du message "*la requête XPath n'a renvoyé aucun résultat*".

Avec l'élément `<date>9 juin 2004</date>`, la même expression renvoie dans oXygen le texte "9 juin 2004".

Dans le cas d'un élément mixte :

```
<article categorie="Le Kiosque">
  <titre>Un rapport encombrant pour Tenet</titre>
  <corps>
    Selon le «<journal>New York Times</journal>», le patron de la CIA a démissionné
avant la publication
    d'un rapport du Sénat très gênant sur les activités de l'agence en Irak.
  </corps>
</article>
```

L'expression `//corps/text()` renvoie dans oXygen deux éléments :

```
Selon le «
```

et

```
le patron de la CIA a démissionné avant la publication d'un rapport du Sénat très gênant
sur les activités de l'agence en Irak.
```

Exercice 9 : Requêtes XPATH.

Les exercices suivants se basent sur la ressource [ANNODIS_me](#)

1. Télécharger l'archive 2_ANNODIS_me_XML.zip contenant les documents au format xml normés selon la TEI-P5 :
 - (a) choisir un fichier et vérifier sa bonne formation
 - (b) vérifier sa validité
 - (c) si le fichier n'est pas indenté correctement, reformatez le en un nouveau fichier xxx_indented.xml pour vous faciliter la lecture
 - (d) ouvrez le navigateur xmllint dans ce fichier
 - (e) lancer une commande xpath permettant d'afficher l'auteur de l'article tel que mentionné dans le teiHeader
 - (f) Afficher la catégorie de l'article tel que mentionné dans le teiHeader
 - (g) trouver le nombre de sections (`div`) en utilisant la fonction `count(chemin xpath)`
 - (h) que signifie cette expression xpath : `//div[count(p)>3]`
 - (i) que signifie cette expression xpath : `//div/p[position()=4]/..`
 - (j) afficher les titres de section
 - (k) afficher les paragraphes ne contenant pas le mot "tout"
2. Télécharger le fichier ANNODIS_CT.xml sur le site redac (corpus)
 - (a) combien y a-t-il de structures ?
 - (b) combien de structures font plus d'un paragraphe ?
 - (c) combien de structures ont plus de 6 expressions coréférentielles ?
 - (d) afficher le texte du premier coréférent `<firstCOREF>` de ces structures

4 Transformations de document XML : XSLT

XSLT est un langage de transformation permettant la transformation d'un document XML vers d'autres formats, notamment des formats d'affichage t.q. HTML.

Une feuille de transformation XSLT est un fichier XML dans lequel sont mentionnées les instructions à appliquer pour transformer un document XML en un nouveau document (nouveau XML, document texte, page html, etc.). Tout comme la norme XPath, le langage XSLT fait partie du protocole de feuilles de style XSL (eXtensible Stylesheet Language).

XPath est dédié au repérage d'éléments dans un document XML (voir section suivante)

XSLT est un langage de programmation qui permet de repérer certains éléments et de les transformer

4.1 XMLNS : la notion de "*namespaces*" (*espaces de nommage*)

Comme indiqué précédemment, les feuilles de style XSLT sont des documents XML qui suivent donc la syntaxe d'un document XML. Cependant, les noms des éléments diffèrent légèrement de ce que l'on a vu pour l'instant, comme on peut le voir avec l'élément racine de base des feuilles XSLT :

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

L'élément racine se caractérise par la forme `<xxx:yyy>` qui indique que l'élément est défini par rapport au "namespace" `xsl` localisé à l'URL <http://www.w3.org/1999/XSL/Transform>.

Actuellement, de plus en plus de documents XML (et plus uniquement les feuilles de transformation XSLT) sont associés à un "namespace" qui correspond à une sorte de vocabulaire XML spécifique. Ce vocabulaire spécifique, appelé espace de nommage, est alors identifié par un URI (*Uniform Resource Identifier*) indiqué dans la balise concernée via un attribut de type `xmlns:XXX` où `xmlns` indique qu'il s'agit d'un namespace et `XXX` indique le *petit* nom à utiliser dans le document XML pour référer à l'espace de nommage. Ex :

L'espace de nom pour un document XML de type XSLT s'indique directement dans l'élément racine du document XSLT. Il correspond à l'URI : <http://www.w3.org/1999/XSL/Transform> et est associé au namespace référencé sous le *petit* nom `xsl`

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform".
```

Ainsi, lorsque l'on fera appel à des éléments (du vocabulaire) propre à cet espace de nommage, on aura un élément de type `xsl:element`.

4.2 Syntaxe d'une feuille de transformation XSLT

Structure de base Comme tout fichier XML bien formé, un fichier XSLT comporte une première ligne déclarant que c'est du xml :

```
<?xml version="1.0" encoding="utf-8"?>
<?xml version="1.0" encoding="iso-8859-1"?>
```

et un élément racine dans lequel on précise la version de la norme xsl utilisée et l'espace de noms utilisé :

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
</xsl:stylesheet>
```

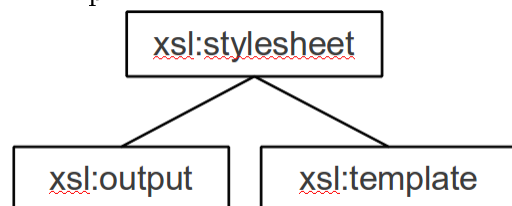
Une première instruction (une fois l'élément racine ouvert) peut indiquer le type de document généré en sortie (exemple pour sortie html) :

```
<xsl:output method="html" version="1.0" encoding="utf-8"/>
<xsl:output method="html" version="1.0" encoding="iso-8859-1"/>
```

L'élément racine est constitué d'autant de motifs (*template* in english) que nécessaires. Un motif indique, pour un élément correspondant (repéré grâce à son adresse XPath), la transformation à effectuer. Tout élément qui ne correspond pas à un motif déclaré sera reproduit tel quel.

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

Toute feuille de style XSLT doit au moins comporter un motif. Classiquement, on définit un motif portant sur la racine du document. L'arbre minimal d'un fichier XSLT est donc le suivant :



Ce qui correspond à :

```
<?xml version="1.0" encoding="utf-8 ou iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" version="1.0" encoding="utf-8 ou iso-8859-1"/>
  <xsl:template match="/">
    CODE PRINCIPAL SANS TEMPLATES QUI PORTE SUR TOUT LE DOC XML
  </xsl:template>
</xsl:stylesheet>
```

Les motifs ont pour attribut l'adresse XPath de l'élément et pour contenu les instructions à effectuer une fois l'élément rencontré.

Attribut match="?"

L'attribut `match` contient une adresse XPath rédigée selon la syntaxe XPath (désignation des chemins et contraintes)

Instructions

Les instructions XSL basiques sont les suivantes :

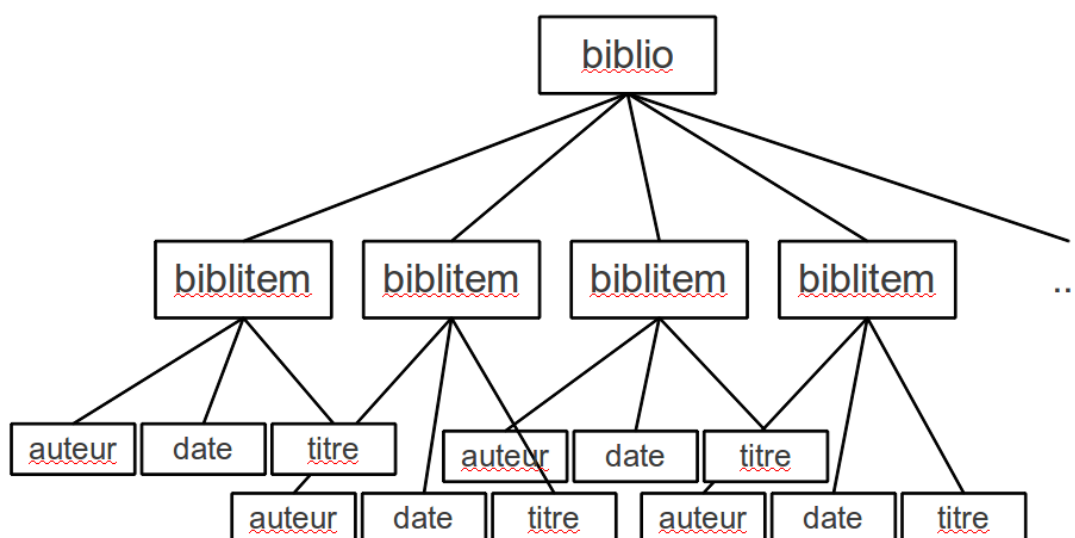
Exemple de feuille de style XSLT (encodée en utf-8) :

Un document XML de type : transformé via la feuille de style :

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

<xsl:template match="/">
  <html>
  <body>
    <h2>Ma bibliographie</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Auteur</th>
        <th>Date</th>
        <th>Titre</th>
      </tr>
    </table>
  </body>
</html>
</xsl:template>
```

<code><xsl:value-of select="nom_element"/></code>	renvoie la valeur de l'élément
<code><xsl:value-of select="@nom_attribut"/></code>	renvoie la valeur de l'attribut
<code><xsl:apply-templates/></code>	poursuivre l'application de motifs à tous les sous-éléments
<code><xsl:apply-templates select="element"/></code>	poursuivre l'application de motifs à tous les sous-éléments sélectionnés
<code><xsl:if test="expr. logique">action </xsl:if></code>	met une condition pour réaliser l'action
<code><xsl:elseif test="expr. logique"> action </xsl:elseif></code>	
<code><xsl:else> action </xsl:else></code>	
<code><xsl:for-each select="element"> action </xsl:for-each></code>	réalise l'action pour chaque élément
<code><xsl:element name="a"> data </xsl:element></code>	créé un élément <i>a</i>
<code><xsl:attribute name="href"> data </xsl:attribute></code>	créé un attribut <i>href</i>



```

</tr>
<xsl:for-each select="biblitem">
  <tr>
    <td><xsl:value-of select="auteur"/></td>
    <td><xsl:value-of select="date"/></td>
    <td><xsl:value-of select="titre"/></td>
  </tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

deviendra une page html (encodée en utf-8) de type :

Ma bibliographie

auteur	date	titre
XXXXX	2222	xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Feuille de style XSLT pour document XML comportant des *namespaces* :

Si le document XML que l'on veut transformer fait appel à un *namespace*, il faut l'indiquer dans la racine du fichier XSLT. Ex : Soit le document XML :

```
<?xml version="1.0" encoding="utf-8"?>
<racine>
  <voc:element xmlns:voc="http://www.vocabulaireSpecifique.fr/">
    <element>text</element>
    <element>text</element>
  </voc:element>
  <voc:element xmlns:voc="http://www.vocabulaireSpecifique.fr/">
    <element>text</element>
  </voc:element>
</racine>
```

Une fichier XSLT associé pourrait être :

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:voc="http://www.vocabulaireSpecifique.fr/">
  <xsl:output method="text" version="1.0" encoding="utf-8"/>
  <xsl:template match="/">
    <xsl:for-each select="voc:element">
      <xsl:value-of select="element"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

4.3 Méthodes de transformation XSLT

Pour appliquer une feuille de style XSLT à un document XML, plusieurs techniques sont envisageables en fonction de l'application :

Transformation via un navigateur web

Une façon d'associer une feuille de style à un document XML dans l'objectif de transformer le document XML en page web (HTML) consiste à ajouter dans le document XML une **information de traitement** indiquant la feuille de style associée. Cette information est à insérer après la déclaration et avant la balise ouvrante de l'élément racine :

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="fichier.xsl" type="text/xsl"?>
<racine>
</racine>
```


Une fois cette ligne insérée, le fichier XML ouvert via un navigateur web (ex : *Firefox*) s'affichera directement avec la feuille de style appliquée (comme une page web). L'inconvénient de cette méthode est que la page HTML issue de la transformation est calculée en mémoire. Cela signifie que vous ne pouvez pas afficher le code source associé à la page affichée, mais uniquement le code source du document XML d'origine.

Exemple : la feuille de style suivante a été créée pour permettre l'affichage HTML des flux RSS via un navigateur WEB.

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

```
<xsl:output method="html" />
<xsl:template match="/rss/channel">
  <h1>PLEINS de titres !!!</h1>
  <ol>
    <xsl:for-each select="item">
      <li><xsl:value-of select="title"/></li>
    </xsl:for-each>
  </ol>
</xsl:template>
</xsl:stylesheet>
```

Exercice 10 : Transformation XSLT via un navigateur web.

Appliquez cette feuille à un des flux RSS téléchargeable depuis la page <http://www.lemonde.fr/rss/> (clic droit sur un des boutons ) et affichez le résultat. Observez également le code source de la page (clic droit sur la page affichée dans le navigateur). Appliquez ensuite cette autre feuille de style (ci-dessous) et observez les différences.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" />
  <xsl:template match="/rss/channel/item">
    <p><xsl:value-of select="title"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Créez une nouvelle feuille de style personnalisée puis appliquez.

4.3.1 Utilisation d'un éditeur XML

Si la feuille de style n'est pas dédiée à un affichage web mais sert à produire un nouveau document quelconque (HTML, XML, TXT, etc.), il faut calculer explicitement ce nouveau document. Pour cela, il faut utiliser un logiciel capable d'appliquer un fichier .xsl à un document .xml. Nous utiliserons ici l'outil de *XSL Transformation* fourni par le paquet *XML Tools* de NotePad++.

Installation : Transformation XSLT via NotePad++.

1. Ouvrir le fichier XML à transformer
2. Vérifier la syntaxe du document XML (Compléments → XML tools → Check XML Syntax Now)
3. Lancer l'outil de transformation XSL (Compléments → XML tools → XSL Transformation)
 - (a) Sélectionner la feuille de style à appliquer (laisser les options vides)
 - (b) Cliquer sur "Transform"
4. Le résultat de la transformation génère un nouveau fichier
5. Si la transformation génère un document HTML, il est possible de le visualiser comme dans un navigateur Web (Compléments → Preview HTML)

Installation : Transformation XSLT via Eclipse.

Tutoriel de l'XSLT processor d'Eclipse Créer un projet, importer les documents xml et xsl, clic droit → run as *XSLT transformation*, exporter l'HTML généré si nécessaire.

Exercice 11 : Transformation XSLT via NotePad++.

Appliquer la feuille de style suivante pour transformer le flux RSS https://www.buzzconcert.com/rss/grandes_villes/concerts/toulouse.xml en fichier texte

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" encoding="utf-8"/>
  <xsl:template match="/rss/channel/item">
    TITRE : <xsl:value-of select="title"/>
    DESC : <xsl:value-of select="description"/>
  </xsl:template>
</xsl:stylesheet>
```

Appliquer la feuille de style suivante pour transformer le fichier xml précédent en fichier html

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8"/>
  <xsl:template match="/rss/channel/item">
    <h1><xsl:value-of select="title"/></h1>
    <p><xsl:value-of select="description"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Essayer ensuite la feuille de style suivante :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8"/>
  <xsl:template match="/rss/channel">
    <h1>PLEINSdetitres!!!</h1>
    <ol>
      <xsl:for-each select="item">
        <li><xsl:value-of select="title"/></li>
      </xsl:for-each>
    </ol>
  </xsl:template>
</xsl:stylesheet>
```

4.3.2 Utilisation d'un script de transformation

Une transformation XSLT peut également être réalisée en utilisant des scripts dédiés, comme le programme **xsltproc**³ qui se lance en ligne de commande.

Installation : Transformation XSLT via xsltproc.

```
xsltproc transform.xsl fichier.xml > fichier.html
```

Exercice 12 : Transformation XSLT.

Créez une feuille de style applicable au fichier ANNODIS_CT.xml permettant d'afficher dans une page html pour chaque structure contenant au moins 6 expressions coréférentielles : son identifiant et le texte des expressions coréférentielles.

3. Installable via la commande : `apt-get install xsltproc`

5 Liens

- tutoriels
 - [w3shools](http://www.w3schools.com/xpath/), ex : <http://www.w3schools.com/xpath/>
 - Cours de Belkacem Chikhaoui (nécessite un minimum de connaissance en programmation Java, enseignée en M2 LITL) : [INF 6450 - Gestion de l'information avec XML](#) (Enseignement à distance de l'Université du Québec : TÉLUQ)
- consortium [TEI](#) et les guidelines de la TEIP5 <http://www.tei-c.org/Guidelines/P5/>