ENC/ XSLT/ Fiche XPath

Jean-Damien Généro

2025-2026

---

1. Les noeuds

Pour XPath, un document XML est un "*tree of nodes*" ("arbre avec des noeuds"). Chaque noeud est une "étape" possible dans une expression XPath. Il y a sept types de noeud :

1.1 *Root node*

- Notation : `/`.
- Sélection : l'ensemble du document ; c'est le seul noeud "orphelin" (il ne peut pas avoir de parent). On parle parfois du *document node*.
- Attention : ne pas confondre avec l'élément racine (*root element*) d'un document XML : `<TEI>` est entièrement contenu dans le *root node* tout en étant le seul *root element* du document.

1.2 *Element nodes*

- Notation : `element()`.
- Séléction : les éléments (i. e. balises) du document XML (entre `<` `>`).

1.3 *Attribute nodes*

- Notation : `attribute()`.
- Sélection : les attributs des balises XML. Le parent direct d'un *attribute node* est forcément un *element node*.

1.4 *Text nodes*

- Notation : `text()`.
- Sélection : le texte d'un *element node* ou d'un *attribute node*.

1.5 *Comment nodes*

- Notation : `comment()`.

- Sélection : les commentaires d'un document XML (notés entre `<!-- -->`).
- Attention : *comment node* sélectionne le texte du commentaire sans les `<!-- -->`.

1.6 *Namespace nodes*

- Notation : *via* l'axe `namespace::`.
- Sélection : les espaces de nom, généralement stockés dans des `xmlns:`. On y accède par l'expression `/tag/namespace::*`.
- Attention :
  - Dans `<TEI xmlns="http://www.tei-c.org/ns/1.0">`, `xmlns` n'est pas un `attribute node` mais un `namespace node`.
  - Très peu utilisés en XSLT.

1.7 *Processing instruction nodes*

- Notation : `processing-instruction()`.
- Séection : les instructions de traitement XML (notées entre `<? ?>`), qui indiquent à une application comment traiter le document XML qu'elle lit. Elles possèdent un nom (`name()`) et une valeur (`string()`).
- Exemple : `<?xml-stylesheet href="exemple.xsl" type="text/xsl" title="Exemple"?>`.
- Attention : ne pas confondre avec l'entête XML (`<?xml version="1.0" encoding="UTF-8"?>` au début du document).

---

2. Les axes

- Les axes (*axis*) permettent de s'émanciper de la direction de base *parent>enfant* d'un chemin XPath ou précisant ou en changeant cette direction.
- Il y a treize axes.
- Ils se notent avec des `::` à la fin ; certains peuvent être abrégés.

2.1 *Child axis*

- Notation : `child::`.
- Sélection : la balise hiérarchiquement inférieure au noeud de contexte.
- Attention :
  - C'est l'axe par défaut, ce qui signie que `/child::TEI/child::teiHeader` est identique à `/TEI/teiHeader`.
  - Les `attribute nodes` et `namespace nodes` sont exclus de cet axe.

2.2 *Parent axis*

- Notation et abréviation : `parent::` et `..` (deux points).
- Sélection : la balise hiérarchiquement supérieure au noeud de contexte.

- Exemples : `//fileDesc/parent::teiHeader` et `//fileDesc/..` sélectionnent le `<teiHeader>` depuis le `<fileDesc>`.

2.3 *Self axis*

- Notation et abréviation : `self::` et `.` (un point).
- Sélection : le noeud de contexte.

2.4 *Attribute axis*

- Notation et abréviation : `attribute::` et `@`.
- Sélection : les attributs du noeud de contexte.
- Attention : si le noeud de contexte est un *attribute node*, il n'y a pas d'axe attribut.
- Exemples : `//titleStmt/title/attribute::level` et `//titleStmt/title/@level` sélectionnent l'attribut `level` de `<title>`.

2.5 *Ancestor axis*

- Notation : `ancestor::`.
- Sélection : tous les noeuds entre la balise hiérarchiquement supérieure au noeud de contexte (= le parent) et le premier du document (= le *root node*).
- Exemple : `//titleStmt/title/ancestor::fileDesc/sourceDesc//author` sélectionne tous les `<author>` dans `<sourceDesc>` depuis `<title>`.

2.6 *Ancestor-or-self axis*

- Notation : `ancestor-or-self::` et `//`.
- Sélection : le noeud de contexte et `ancestor axis`.

2.7 *Descendant axis*

- Notation et abréviation : `descendant::`.
- Sélection : tous les enfants du noeud de contexte, et leurs enfants, et ainsi de suite.
- Exemple : `//descendant::sourceDesc`.
- Attention : les `attribute nodes` et `namespace nodes` ne sont pas pris en compte par cet axe.

2.8 *Descendant-or-self axis*

- Notation : `descendant-or-self::`.
- Sélection : le noeud de contexte et `descendant axis`
- Attention : les `attribute nodes` et `namespace nodes` ne sont pas pris en compte par cet axe.

2.9 *Preceding-sibling axis*

- Notation : `preceding-sibling::`.
- Sélection : tous les noeuds qui ont le même parent que le noeud de contexte et qui sont avant lui dans l'arbre.
- Exemples :
  - `//sourceDesc/preceding-sibling::titleStmt` sélectionne le `<titleStmt>` depuis le `<sourceDesc>` ;
  - `//sourceDesc/preceding-sibling::*` sélectionne le `<titleStmt>` et le `<publicationStmt>`.

2.9 *Following-sibling axis*

- Notation : `following-sibling::`.
- Sélection : tous les noeuds qui ont le même parent que le noeud de contexte et qui sont après lui dans l'arbre.
- Exemple : `//publicationStmt/following-sibling::sourceDesc` sélectionne le `<sourceDesc>` depuis le `<publicationStmt>`.

2.10 *Preceding axis*

- Notation : `preceding::`.
- Sélection : tous les noeuds qui précèdent le noeud de contexte, peu importe leur parent.
- Attention : les `ancestor`, les `attribute` et les `namespace` ne sont pas pris en compte.

2.12 *Following axis*

- Notation : `following::`.
- Sélection : tous les noeuds qui suivent le noeud de contexte, peu importe leur parent.
- Attention : les `ancestor`, les `attribute` et les `namespace` ne sont pas pris en compte.

2.13 *Namespace axis*

- Notation : `namespace::`.
- Sélection : les noms de domaine du noeud de contexte.
- Attention : ne fonctionne pas avec un *attribute node*.

---

3. Fonctions XPath

XPath possède des fonctions prédéfinies :

- Elles sont reconnaissables par leurs `()` finales ;

- Elles peuvent avoir un ou plusieurs arguments, ou ne pas en avoir du tout ;
- Dans XSLT, elles peuvent être utilisées dans les `@match` ou les `@select` ;
- Elles peuvent retourner : une valeur booléenne (`true` ou `false`), un nombre, une chaîne de caractères (`string`) ou une liste de noeuds (`nodeset`) ;
- Les opérateurs suivants sont utilisables avec les fonctions et prédicats :
  - Opérateurs numériques : `+`, `-`, `*`, `div` (division), `mod` (modulo).
  - Opérateurs booléens : `<`, `<=`, `>`, `>=`, `=`, `!=`, `and`, `or`.

Sélection de fonctions :

3.1 `concat()`

- `concat(string1, string2, ...)` : retourne la concaténation de `string1` et `string2` (et plus).

3.2 `upper-case()` et `lower-case()`

- `upper-case(string)` : retourne `string` en haut-de-casse ;
- `lower-case(string)` : retourne `string` en bas-de-casse.

3.3 `translate()` et `replace()`

- `translate()` prend une liste de caractères en entrée et remplace chaque caractère par son équivalent dans une deuxième liste (= remplace caractère par caractère).
  - `translate(string, 'abc', 'ABC')` : retourne `string` avec :
    - `a` remplacé par `A` ;
    - `b` remplacé par `B` ;
    - `c` remplacé par `c`.
  - *NB:* le troisième paramètre n'a pas besoin d'être de la même taille que le deuxième. Avec `translate(string, 'abc', '')`, tous les `a`, les `b` et les `c` seront enlevés de `string`.
  - *NB:* ne fonctionne pas avec des regex.
  - Ce n'est pas l'équivalent d'un "rechercher/remplacer".
- `replace(string, 'abc', 'ABC')` : retourne `string` avec `abc` remplacé par `ABC`.
  - Peut être considéré comme l'équivalent d'un "rechercher/remplacer".
  - *NB:* fonctionne avec des regex.
  - Attention: ne fonctionne qu'à partir de XSLT version 2.

3.4 `contains()`

- `contains(string1, string2)` : retourne `true` ou `false` selon que `string1` contienne (`true`) ou ne contienne pas `string2` (`false`).

3.5 `starts-with()` et `ends-with()`

- `starts-with(string1, string2)` : retourne `true` ou `false` selon que `string1` commence (`true`) ou ne commence pas par `string2` (`false`) ;
- `ends-with(string1, string2)` : retourne `true` ou `false` selon que `string1` se termine (`true`) ou ne se termine pas par `string2` (`false`) ;

3.6 `not()`

- `not(arg)` : évalue `arg` et retourne la valeur opposée.
  - `\\title[not(@level = 'm')]` renvoie tous les `<title>` dont le `@level` n'a pas pour valeur `m` ainsi que ceux qui n'ont pas de `@n`.
  - *Attention : ce n'est pas l'équivalent de l'opérateur booléen `!=`. `\\title[@level != 'm')]` renvoie tous les `<title>` qui ont un `@level` dont la valeur n'est pas `m`.*
  - Peut être combiné avec `contains()` sous la forme de `not(contains(..., ...))`.

3.7 `last()`, `position()` et `count()`

- *NB : dans XPath, la numérotation commence à `1`.*
- `last( )` : évalue la taille d'un ensemble de balises.
  - Exemple d'utilisation : appliquer une règle à la dernière balise d'un ensemble. Ainsi, `//div/p[last()]` renvoie tous les `<p>` en dernière position dans les `<div>`.
- `position( )` : évalue la position d'une balise au sein de l'ensemble de balises auquel s'applique la règle courante.
  - Exemple d'utilisation : atteindre une balise en fonction de son emplacement. Ainsi, `//div/p[position() = 5]` renvoie tous les `<p>` en cinquième position dans les `<div>`.
  - `//div/p[position() = 5]` et `//div/p[5]` sont deux expressions identiques.
  - *Attention : `position( )` ne permet pas de déterminer la position de la balise par rapport à son parent.*
- `count(arg)` : évalue le nombre de `node` et retourne ce nombre.
  - Le résultat est identique à `last()`, la différence étant que `count()` prend une expression XPath en argument.
  - Exemple d'utilisation : connaître le nombre total d'élément au sein d'un parent (comme le nombre de `<p>` au sein d'une `<div>`).

3.8 `tokenize()`

- `tokenize(string, delimiter)` : divise la chaîne `string` en fonction d'un `delimiter`.

– Nouveauté de XSLT 2.
– Exemple d'utilisation : manipuler les valeurs multiples d'un attribut.
  Si `@type="foo bar"`, `tokenize(@type, ' ')` retour `foo` et `bar`.