

CIHAM (UMR 5648), CNRS

Introduction à XSLT

Biblissima+, scripts et manuscrits

Ariane Pinche
ariane.pinche@cnrs.fr

Biblissima+, scripts et manuscrits, 5-7 mai 2025

- 1 XSLT et son environnement
- 2 Principes de fonctionnement de XSL
- 3 Exercice final
- 4 Conclusion

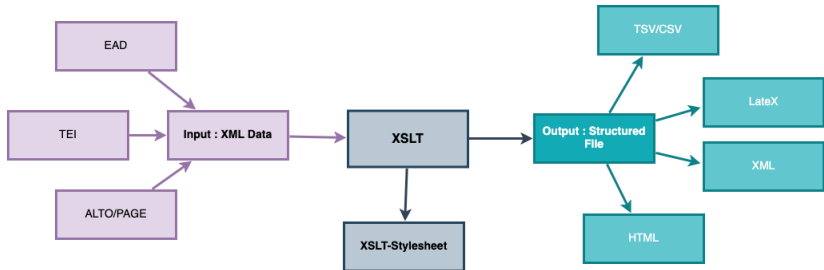


Figure: Représentation de l'environnement XSLT

Définition : « XSLT (extensible Stylesheet Language Transformations) est un langage de programmation fonctionnel utilisé pour spécifier comment un document XML est transformé en un autre document qui peut, mais qui n'est pas nécessairement, un autre document XML. Un processeur XSLT lit un arbre XML en entrée et une feuille de style XSL et produit un arbre résultat en sortie. »
Elliott Rusty Harold, W. Scott Means, Philippe Ensarguet [et al.],
XML en concentré, Paris, O'Reilly, 2005, p. 519.

- Langage de transformation de documents XML
- Permet de produire **tout type de document** à partir d'un XML :
 - HTML bien formé (avec balises fermées)
 - Autre XML (ex. ALTO → TEI)
 - Formats non-XML (ex. TSV, LaTeX)
- Langage XML normalisé par le W3C (bien formé)
- **Principe central** : traduire des éléments XML en autre chose

- **XSLT 1.0** (1999) – supporté par les navigateurs web :
<https://www.w3.org/TR/xslt>
- **XSLT 2.0** (2007) – supporté par les processeurs avancés (ex. Saxon) :
<https://www.w3.org/TR/xslt20/>
- **XSLT 3.0** (2012) :
<https://www.w3.org/TR/xslt-30/>

- **XPath** : sélection d'éléments XML (utilisé par XSLT, XQuery, etc.)
<https://www.w3.org/TR/xpath/>
- **XSL-FO** (XSL Formatting Objects) : production dynamique de PDF
<https://www.w3.org/TR/xsl/>
- **XQuery** : langage de requête XML, adapté à l'extraction de données
<https://www.w3.org/TR/xquery/>

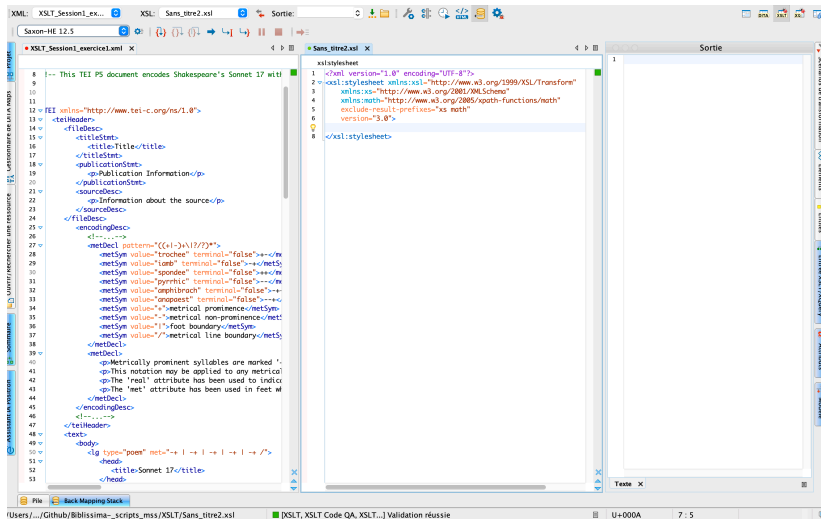


Figure: Interface d'Oxygen

- ① XSLT et son environnement
- ② **Principes de fonctionnement de XSL**
- ③ Exercice final
- ④ Conclusion

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

- Éléments obligatoires :
 - Une déclaration XML au début du fichier
 - Un élément racine *stylesheet*
 - Une déclaration de l'espace de nom

Démonstration dans Oxygen

- Lorsqu'aucune règle explicite n'est trouvée, XSLT applique des **règles par défaut** : seul le **contenu textuel est copié** dans la sortie, **les balises ne le sont pas**.
- L'arbre XML est parcouru de façon **récursive** : chaque nœud est traité de la **racine à ces descendants**.
- **L'ordre des règles n'a aucune importance** : la règle la plus **spécifique** est toujours prioritaire.

À quoi sert Xpath avec XSLT ?

- Naviguer dans l'arbre pour sélectionner des éléments ou des attributs

Un peu d'application

① Ouvrir un fichier xsl dans Oxygen

② Écrire dans la feuille :

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

① À partir du fichier XML *XSLT_Session1_exercice1.xml* et à l'aide de la règle copy-of, écrivez l'expression XPath dans l'attribut *match* permettant de copier uniquement l'élément **lg** ayant un attribut **couplet**.

② Copier maintenant le ou les éléments **lg** ayant un attribut **quatrain** et en position 1

Pour donner des instructions XSL, on utilise des templates (règles) via l'élément :

```
<xsl:template>
```

On utilise l'attribut `match` pour sélectionner un élément de l'arbre XML :

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

On peut ajouter dans la règle du texte :

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

Ici, il y avait mon élément

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

ou

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

```
  <xsl:text>Ici, il y avait mon élément</xsl:text>
```

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

... mais aussi des éléments ou des nouvelles règles

Démonstration dans Oxygen

Méthode 1 :

```
<xsl:template match="mon_element_xml">  
  <p>Ici, il y avait mon élément</p>  
</xsl:template>
```

Méthode 2 :

```
<xsl:template match="mon_element_xml">  
  <xsl:element name="nom_element_a_creer">  
    Ici, il y avait mon élément  
  </xsl:element>  
</xsl:template>
```

Démonstration dans Oxygen

Méthode 1 :

```
<xsl:template match="mon_element_xml">  
  <p type="valeur_attribut">Ici, il y avait mon élément</p>  
</xsl:template>
```

Ou avec XPath :

```
<xsl:template match="mon_element_xml">  
  <p type="{./chemin_Xpath}">Ici, il y avait mon élément</p>  
</xsl:template>
```


Méthode 2 :

```
<xsl:template match="mon_element_xml">
  <xsl:element name="nom_element_a_creer">
    <xsl:attribute name="nom_attribut_a_creer">
      <xsl:text>valeur_attribut</xsl:text>
    </xsl:attribute>
    <xsl:text>Ici, il y avait mon élément</xsl:text>
  </xsl:element>
</xsl:template>
```

Démonstration dans Oxygen

value-of Insère la valeur textuelle d'un noeud sélectionné par une expression XPath dans la sortie.

```
<xsl:template match="mon_element_xml">  
  <xsl:value-of select="chemin_Xpath"/>  
</xsl:template>
```

Démonstration dans Oxygen

L'élément *apply-templates* permet d'appliquer les règles définies aux éléments enfants de l'élément sélectionné dans l'attribut match.

```
<xsl:template match="mon_element_xml">  
  <xsl:element name="nom_element_a_creer">  
    <xsl:apply-templates/>  
  </xsl:element>  
</xsl:template>
```

Démonstration dans Oxygen

L'élément *copy-of* copie le nœud sélectionné et tous ses enfants, attributs et descendants.

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

Démonstration dans Oxygen

L'élément *copy* copie le nœud courant, sans ses enfants ni ses attributs.

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

Démonstration dans Oxygen

L'élément *output* permet de paramétrer le format de sortie des données, l'encodage des caractères, ainsi que l'indentation du document de sortie.

```
<xsl:output method="xml" indent="yes" encoding="UTF-8"/>
```

Démonstration dans Oxygen

- `xsl:number` produit un nombre formaté.
- Attributs :
 - `count` : motif à compter
 - `level` : profondeur (single, multiple, any)
 - `format` : format de numérotation

```
<xsl:template match="mon_element_xml">  
  <xsl:number count="element_a_compter" level="type_numerotation"  
    format="format_numerotation"/>  
</xsl:template>
```

Démonstration dans Oxygen

- 1 XSLT et son environnement
- 2 Principes de fonctionnement de XSL
- 3 Exercice final
- 4 Conclusion

Objectif : Reproduire un fichier TEI en numérotant automatiquement les vers et les strophes.

- Créez une nouvelle feuille XSLT.
- Paramétrez correctement le préambule en déclarant l'espace de nom TEI et en configurant son utilisation par défaut dans XPath.
- Reproduisez la structure du document TEI d'entrée, tout en ajoutant une numérotation automatique pour les vers et les strophes.
- Utilisez les éléments XSLT étudiés lors de la session 1 :
- Proposez plusieurs formats de numérotation (romaine, décimale, hiérarchique, etc.).
- N'oubliez pas d'ajouter l'attribut `type` sur les éléments **lg**.

- 1 XSLT et son environnement
- 2 Principes de fonctionnement de XSL
- 3 Exercice final
- 4 Conclusion

- Une feuille XSLT se compose de **règles (template)** qui utilisent **l'attribut match** pour cibler les éléments spécifiques de l'arbre XML à transformer.
- L'arbre XML est parcouru de manière **récursive** : chaque nœud est traité, de la **racine jusqu'aux descendants**.
- XPath est utilisé pour naviguer dans l'arbre XML et sélectionner les données souhaitées.
- Il est possible de générer des éléments (element), des attributs (attribute) ou d'insérer du texte (text) dans le document transformé.
- L'élément apply-templates permet d'appliquer des règles aux éléments enfants d'un nœud à l'intérieur d'une règle template