

# **Séance 11 : Personnaliser son ODD**

# Personnalisation du schéma

Pour personnaliser son schéma, il existe 4 manipulations principales :

- Ajouter des éléments ;
- Supprimer des éléments ;
- Changer des éléments ;
- Personnaliser les attributs et les valeurs d'attribut d'un élément.

## Suppression d'un élément

1- Suppression simple d'un élément avec @mode="delete"

- `<elementSpec ident="head" mode="delete"/>`

2- Suppression d'un élément ou une classe dans un module avec @except

- `<moduleRef key="core" except="head"/>`

3- Non insertion dans un module ou une classe avec @include

- `<moduleRef key="core" include="p author title l lg"/>`

## Modification d'un élément

Modification d'un élément avec @mode="change"

```
<elementSpec ident="lg" mode="change">
  <attList>
    <attDef ident="type" mode="change">
      <valList mode="add" type="closed">
        <valItem ident="quatrain"/>
        <valItem ident="sizain"/>
        <valItem ident="sonnet"/>
        <valItem ident="tercet"/>
      </valList>
    </attDef>
  </attList>
</elementSpec>
```

- Modifier dans l'ODD du texte de Verlaine l'élément `<l>` en lui ajoutant un attribut "n" obligatoire.

# Typage des données

Utilisation dans l'élément de définition d'une balise ou d'un attribut

`<dataType>` et de son enfant `<dataRef>` avec l'attribut `@key` si on pointe vers un type de données défini par la TEI et un attribut `@name` si on pointe vers un type de données défini dans XML schéma ou RelaxNG :

```
<attDef ident="n" mode="change">  
  <datatype>  
    <dataRef key="teidata.count"/>  
  </datatype>  
</attDef>
```

teidata.count =

```
<content>  
  <dataRef name="nonNegativeInteger"/>  
</content>
```

Liste des types de données TEI : <https://www.tei-c.org/release/doc/tei-p5-doc/fr/html/REF-MACROS.html>

# Addition d'un élément

- Déclaration d'un élément dans le `<moduleRef>` avec `@include`
  - `<moduleRef key="textstructure" include="TEI text body front"/>`
- Création d'un nouvel élément qui n'appartient pas au domaine TEI :

```
<elementSpec ident="alexandrin"
              mode="add"
              ns="http://www.example.org/ns/nonTEI">
  <classes>
    <memberOf key="model.lLike"/>
    <memberOf key="macro.paraContent"/>
  </classes>
  <content>
    <textNode/>
  </content>
</elementSpec>
```

- Déclarer dans l'ODD du texte de Verlaine un attribut "subtype" (non TEI) non obligatoire avec une liste de valeurs close : "rime A

## Exercice en autonomie – Créer l'ODD de Lucain à partir de oddbyexample.

- Créer l'ODD de Lucain à partir de oddbyexample
- Lier votre schéma à votre XML (vérifier la validation)
- Opérer les modifications suivantes à la main
- Ajouter l'attribut subtype sur les rdg et lem
- Modifier les valeurs de l'attribut type des `<rdg>` en faisant une liste fermée comprenant les valeurs :
  - graphic
  - semantic
- N'autoriser que les nombres entiers dans les valeurs de l'attribut @n de l'élément l

# Documenter son ODD

L'ODD comporte :

- un élément racine TEI ;
- un teiHeader ;
- un élément text ;

L'élément body peut contenir des div à niveau (div1, div2, etc.)

On peut structurer sa documentation dans une première div1 et placer ses spécifications dans une autre div1.



L'intégralité des spécifications XML est englobée dans un **schemaSpec**

- Les déclarations sont organisées grâce aux éléments suivants (liste non exhaustive) :
  - **moduleSpec** (spécification de module) documente la structure, le contenu et les fonctions d'un module.
  - **moduleRef** (référence de module) référence un module qui doit être incorporé dans un schéma.
  - **elementSpec** (spécification d'élément) documente la structure, le contenu et l'emploi d'un élément.
  - **classSpec** (spécification de classe) contient des informations de référence pour une classe d'éléments TEI, c'est-à-dire un groupe d'éléments qui figurent ensemble dans des modèles de contenu ou qui partagent un attribut commun, ou qui ont l'un et l'autre.

# 1-Les éléments structurants de la documentation

Dans l'introduction (première div1), au sein d'un texte rédigé :

- **specList** (liste de spécification) marque l'endroit où insérer une liste de descriptions dans le texte documentaire ;
- **specDesc** (specification description) indique qu'une description de l'élément particulier ou de la classe particulière doit être incluse à ce point dans un document ;
- **egXML** et **@xmlns="http://www.tei-c.org/ns/Examples"** permettent d'insérer des exemples en XML dans sa documentation ;

## Exemple :

```
<head> Le fileDesc </head>
```

```
<p> Le <gi> fileDesc </gi> comporte lui-même :
```

```
<specList>
```

```
<specDesc key="titleStmt"/>
```

```
[...]
```

```
</specList>
```

Le <gi> sourceDesc </gi> contient toutes les informations nécessaires sur le manuscrit de base, C <hi

```
rend="exp"> 1 </hi> `` <note> Le sigle correspond au manuscrit 412  
de la Bibliothèque Nationale de France </note> . </p>
```

```
<egXML xmlns="http://www.tei-c.org/ns/Examples">
```

```
<msIdentifier>
```

```
<country> Paris </country>
```

```
<settlement> Bibliothèque nationale de France </settlement>
```

```
[...]
```

```
</egXML>
```

## Dans les déclarations d'éléments

- Dans **elementSpec** ou **attDef**.

**gloss** (glose) identifie une expression ou un mot utilisé pour fournir une glose ou une définition.

**desc** (description) contient une courte description de l'objet documenté par son élément parent, qui comprend son utilisation prévue, son but, ou son application là où c'est approprié.

```
<elementSpec ident="lem" mode="change">
```

```
<gloss>Lemme</gloss>
```

```
<desc>Permet de signaler la leçon choisie dans le  
texte édité.</desc>
```

```
[...]
```

```
</elementSpec>
```

## 2- Syntaxe des éléments XML à signaler dans sa documentation

**att** (attribut) contient le nom d'un attribut apparaissant dans le courant du texte.

**gi** (identifiant générique) contient le nom d'un élément.

**tag** (balise) le contenu d'une balise ouvrante ou fermante, avec éventuellement des spécifications d'attributs, mais à l'exclusion des caractères marquant l'ouverture et la fermeture de la balise.

**val** (valeur) contient une seule valeur d'attribut.

Exemple :

Le corpus présente trois cas de figure. Dans le premier cas, la ponctuation originale est supprimée dans l'édition normalisée. L'ajout de l'attribut `<att> type </att>` de valeur `<val> orig </val>` sur l'élément `<gi> pc </gi>` signale que le signe est issu de la ponctuation du manuscrit et qu'il ne doit pas apparaître dans la version normalisée.

## Exercice

- Créer la documentation de l'ODD de Lucain.
  - Insérer dans votre documentation au moins un exemple
  - Modifier la documentation d'un élément en réécrivant sa description pour la faire correspondre au projet.
- Générer vos guidelines en HTML

## **II- Personnaliser son ODD, niveau 2**



## 1-Définir les modalités d'apparition d'un élément

La règle définissant une séquence apparaît directement en dessus de la balise ouvrante de l'**elementSpec** dans un élément **content**.

La séquence est contenue dans un élément **sequence** avec un attribut **preserveOrder** qui permet de spécifier si l'ordre de déclaration des éléments de la séquence est signifiant.

Chaque élément est appelé à l'aide d'un **elementRef** et d'un attribut **key** qui permet de donner le nom de l'élément. On peut également définir les modalités d'apparition des éléments de la séquence à l'aide des attributs **minOccurs** et **maxOccurs**.

## Exemple

```
<elementSpec ident="div" mode="change">
  <content>
    <sequence preserveOrder="true">
      <elementRef key="head" minOccurs="1" maxOccurs="1"/>
      <elementRef key="p" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
  </content>
</elementSpec>
```

NB : Pour autoriser du texte comme contenu, on peut ajouter dans la séquence : `<textNode/>`

Il est également possible de raffiner ses séquences avec l'élément

`<alternate>`

*Exemple : définition du contenu de `<choice>` :*

```
<alternate>
  <sequence>
    <elementRef key="sic"/>
    <elementRef key="corr"/>
  </sequence>
  <sequence>
    <elementRef key="orig"/>
    <elementRef key="reg"/>
  </sequence>
  <sequence>
    <elementRef key="abbr"/>
    <elementRef key="expan"/>
  </sequence>
</alternate>
```

## 2-contraindre des valeurs d'attributs, des enchaînements en fonction du contexte (schematron)

- La contrainte est introduite par un élément `<constraintSpec>`.  
Le langage utilisé est déclaré dans l'attribut **scheme="schematron"**, la règle est nommée à l'aide de l'attribut *ident*.
- La règle est contenue dans une balise `<constraint>`.

NB : Attention à bien déclarer le nom de domaine *schematron* dans le préambule : `xmlns:s="http://purl.oclc.org/dsdl/schematron"`

## Exemple

```
<constraintSpec ident="reforkeyorname" scheme="schematron"
  <constraint>
    <s:assert test="@ref or @key or @name">One of the
      attributes 'name',
      'ref' or 'key' must be supplied</s:assert>
    </constraint>
  </constraintSpec>
```

**s:assert** permet la vérification de l'existence de la contrainte rédigée en Xpath.

## Ajouter un contexte

```
<constraintSpec ident="subclauses"  
  scheme="schematron">  
  <constraint>  
    <s:rule context="tei:div">  
      <s:assert test="count( tei:div ) != 1">  
        if it contains any subdivisions,  
        a division must contain at least two of them  
      </s:assert>  
    </s:rule>  
  </constraint>  
</constraintSpec>
```

**s:rule** permet d'ajouter un contexte à l'application de **s:assert**.

Contraindre l'activation d'un élément ou d'un attribut en fonction d'un contexte donné

```
<constraintSpec ident="fromTo" scheme="schematron">  
  <constraint>  
    <s:rule context="tei:app[@type='structure']">  
      <s:assert test="@from and @to">  
        The beginning and the endpoint of the  
        lemma have to be identify/  
      </s:assert>  
    </s:rule>  
  </constraint>  
</constraintSpec>
```

Contraindre le type de contenu d'une valeur d'attribut :

```
<constraintSpec ident="fromType" scheme="isoschematron">
  <constraint>
    <s:rule context="tei:app[@from]">
      <s:assert test="matches(@from, '^#w\d+$')">
        @from='#w+nb'` </s:assert>
      </s:rule>
    </constraint>
  </constraintSpec>
```



# Exercice

Reprendre le fichier XML de Lucain,

## 1-Ajouter des règles à votre XML

- Paramétrer l'ordre de la séquence des éléments de l'apparat de telle sorte à ce que le lemme soit déclaré avant les leçons.
- rendre obligatoire la présence d'un seul `<lem>` ;
- rendre obligatoire la présence d'une ou plusieurs leçons ;
- Écrire une règle schematron pour que les valeurs @n de `<l>` se suivent de telle sorte que : `number(@n) = number(preceding-sibling::tei:l[1]/@n) + 1` quand le vers n'est pas en première position (reprendre le cours de Xpath si nécessaire);
- Générer votre schéma RelaxNG (syntax XML) et l'associer à votre fichier XML.