



# Séance 12

# Les schémas

# XML

Définir des règles de validation pour son  
encodage

A decorative graphic on the left side of the slide consisting of two overlapping squares. The bottom-left square is a dark blue, and the top-right square is a lighter blue, creating a cross-like shape.

# **Restreindre le contenu d'élément**

Comment préciser ce que peut contenir un  
élément dans un encodage

# elementSpec

```
<elementSpec ident="elementName" mode="change">
  <gloss>Explicitation de l'élément</gloss>
  <desc>Description plus longue de l'élément</desc>
  <content>
    <!-- DÉFINITION DU CONTENU DE L'ÉLÉMENT -->
  </content>
  <constraintSpec ident="constName" scheme="schematron">
    <!-- RÈGLE SCHEMATRON -->
  </constraintSpec>
  <attList>
    <!-- LISTE DES ATTRIBUTS DE L'ÉLÉMENT -->
    <attDef ident="attName" mode="change">
      <desc>Description de l'attribut</desc>
      <valList><!-- LISTE DE VALEURS --></valList>
    </attDef>
  </attList>
</elementSpec>
```

# Déclarer du contenu

<content> apparaît toujours avant <attList> et <constraintSpec> dans l'<elementSpec> et il peut contenir :

- Une **référence** à un élément / classe / module
- <**alternate**> qui définit une alternative d'éléments
- <**sequence**> qui définit une succession d'éléments
- <**empty**/> ou <**textNode**/>

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div contient un élément head
  </desc>
  <content>
    <elementRef key="head"/>
  </content>
</elementSpec>
```

*Il supprime le contenu défini par la TEI mais pas les règles schématron*

# Quelques références disponibles

## **moduleRef**

Référence à un module à inclure dans le schéma

## **macroRef**

Référence à une macro à inclure dans un modèle de contenu

## **elementRef**

Référence à un élément à inclure comme contenu possible d'un élément

## **classRef**

Référence à un modèle de contenu à inclure comme contenu possible d'un élément

# Définir un contenu textuel / vide

```
<elementSpec ident="div" mode="change">  
  <content>  
    <sequence>  
      <textNode/>  
      <elementRef key="persName"/>  
    </sequence>  
  </content>  
</elementSpec>
```

```
<elementSpec ident="lb" mode="change">  
  <content>  
    <empty/>  
  </content>  
</elementSpec>
```

# Dénombrer le contenu

La valeur par défaut de @minOccurs et @maxOccurs est de **1** et peut avoir pour valeur zero\_ou\_plus | “unbounded”

 Sauf mention contraire, **tous les éléments** déclarés dans content doivent apparaître.

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div contient 1 ou plusieurs éléments p
  </desc>
  <content>
    <elementRef key="p" minOccurs="1" maxOccurs="unbounded" />
  </content>
</elementSpec>
```

# Déclarer du contenu optionnel

Pour définir un contenu optionnel, indiquer **@minOccurs="0"**

Il est ainsi possible de définir plusieurs contenus possibles pour un même élément.

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div peut contenir un unique élément
    <head> et/ou un unique élément <p>
  </desc>
  <content>
    <elementRef key="head" minOccurs="0"/>
    <elementRef key="p" minOccurs="0"/>
  </content>
</elementSpec>
```



# Définir une alternative

L'élément `<alternate>` permet de définir un ensemble cohérent de contenus pouvant apparaître dans un élément.

⚠ Sauf indication contraire, **seulement un des contenus** de `<alternate>` peut apparaître.

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div contient un <head> ou un <p>
  </desc>
  <content>
    <alternate>
      <elementRef key="head"/>
      <elementRef key="p"/>
    </alternate>
  </content>
</elementSpec>
```

# Changer les occurrences

Avec **alternate**[@maxOccurs="unbounded"], tous les éléments déclarés peuvent apparaître autant de fois qu'on veut

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div contient au moins un <head>
    et/ou au moins un <p>
  </desc>
  <content>
    <alternate maxOccurs="unbounded">
      <elementRef key="head"/>
      <elementRef key="p"/>
    </alternate>
  </content>
</elementSpec>
```

# Définir une séquence de contenu

L'élément <sequence> permet de définir une suite précise de contenus dont l'**ordre est significatif**

```
<elementSpec ident="div" mode="change">
  <desc xml:lang="fr">
    L'élément div contient une ou plusieurs
    séquences de <head> suivi de un ou
    plusieurs <p>
  </desc>
  <content>
    <sequence maxOccurs="unbounded">
      <elementRef key="head"/>
      <elementRef key="p" maxOccurs="unbounded"/>
    </sequence>
  </content>
</elementSpec>
```

# Imbriquer les séquences

Les séquences peuvent être démultipliées et imbriquées

```
<elementSpec ident="div" mode="change">
  <content>
    <sequence> <!-- 1x <head> puis <p> -->
      <elementRef key="head"/>
      <elementRef key="p"/>
    </sequence>
    <sequence minOccurs="0"> <!-- séquence optionnelle -->
      <alternate> <!-- <name> ou <persName> -->
        <elementRef key="name"/>
        <elementRef key="persName"/>
      </alternate> <!-- puis un élément de pLike -->
      <classRef key="model.pLike"/>
    </sequence>
  </content>
</elementSpec>
```

# Définir une alternance de contenu

```
<content>
  <alternate>
    <sequence> <!-- <street> puis <placeName> puis ... -->
      <elementRef key="street"/>
      <elementRef key="placeName"/>
      <elementRef key="postCode"/>
      <elementRef key="country"
        minOccurs="0"
        maxOccurs="1"/>
    </sequence>
    <!-- OU <addrLine> de 2 à 4x -->
    <elementRef key="addrLine"
      minOccurs="2"
      maxOccurs="4"/>
  </alternate>
</content>
```

# Comprendre une séquence de contenu des *Guidelines* : floatingText

```
<content>
  <sequence>
    <!-- peut contenir des éléments de la classe model.global -->
    <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>

    <!-- peut contenir une séquence de <front> et des éléments de model.global -->
    <sequence minOccurs="0">
      <elementRef key="front"/>
      <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>

    <!-- DOIT contenir <body> ou <group> -->
    <alternate>
      <elementRef key="body"/>
      <elementRef key="group"/>
    </alternate>

    <!-- peut contenir une séquence de <back> et des éléments de model.global -->
    <sequence minOccurs="0">
      <elementRef key="back"/>
      <classRef key="model.global" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </sequence>
</content>
```

# Exercice

Définir un <content>



# Consigne

- Reprendre l'[ODD pour sonnetTEI.xml](#)
- Définir un <content> pour contraindre le contenu d'un <lg> à être soit :
  - un enchaînement d'un titre puis des strophes ;
  - deux strophes ;
  - quatre vers.
- Générer le schéma RelaxNG et l'associer au fichier TEI



A decorative graphic on the left side of the slide consisting of two overlapping squares. The bottom-left square is a dark blue, and the top-right square is a lighter blue, creating a cross-like shape.

# Règles de validation

La syntaxe Schematron

# XML namespace

Utilisé pour **éviter les conflits de noms d'éléments** quand plusieurs espaces de noms sont utilisés ensemble. Tous les éléments utilisant le *namespace* doivent être contenus par un élément où est spécifié l'URI du *namespace* dans les attributs.

## Sans préfixe

```
<TEI xmlns="ns_URI">
  <teiHeader/>
  <text/>
</TEI>
```

## Avec préfixe

```
<a:div xmlns:a="URI_a">
  <a:h1/>
  <a:text/>
</a:div>

<b:div xmlns:b="URI_b">
  <b:h1/>
  <b:text/>
</b:div>
```

## Double

```
<root xmlns:a="URI_a"
       xmlns:b="URI_b">
  <a:div>
    <a:h1/>
    <a:text/>
  </a:div>

  <b:div>
    <b:h1/>
    <b:text/>
  </b:div>
</root>
```

# Règle schematron et *namespace*

Les règles schematron n'utilisent pas le même *namespace* que le reste de l'ODD qui utilise l'espace de nom TEI : il faut donc déclarer un @xmlns dans l'élément racine et préfixer les règles schematron.

Les chemins renseignés dans la règle schematron doivent être **préfixés avec le namespace tei** (qui n'a pas besoin d'être déclaré).

```
<TEI xmlns="http://www.tei-c.org/ns/1.0"
      xmlns:s="http://purl.oclc.org/dsdl/schematron">

  [...]
  <constraint>
    <s:assert test="tei:div">[...]</s:assert>
  </constraint>

</TEI>
```

# elementSpec

```
<elementSpec ident="elementName" mode="change">
  <gloss>Explicitation de l'élément</gloss>
  <desc>Description plus longue de l'élément</desc>
  <content>
    <!-- DÉFINITION DU CONTENU DE L'ÉLÉMENT -->
  </content>
  <constraintSpec ident="constName" scheme="schematron">
    <!-- RÈGLE SCHEMATRON -->
  </constraintSpec>
  <attList>
    <!-- LISTE DES ATTRIBUTS DE L'ÉLÉMENT -->
    <attDef ident="attName" mode="change">
      <desc>Description de l'attribut</desc>
      <valList><!-- LISTE DE VALEURS --></valList>
    </attDef>
  </attList>
</elementSpec>
```

# Contraindre un élément

La règle est contenue dans un élément `<constraintSpec>`. Il est obligatoire de lui donner un nom avec `@ident`.

`<s:assert>` permet de définir :

- un **test** à respecter en [XPath](#) avec `@test` (~prédicats)
- un **message d'erreur** si la contrainte n'est pas respectée

```
<elementSpec ident="div" mode="change">
  <constraintSpec ident="attReq" scheme="schematron">
    <constraint>
      <s:assert test="@n">
        L'élément div doit avoir un attribut @n
      </s:assert>
    </constraint>
  </constraintSpec>
</elementSpec>
```

# assert / report

`<assert>` renvoie une erreur si la condition dans `@test` est fausse

`<report>` renvoie une erreur si la condition dans `@test` est vraie

Un seul `assert/report` par `constraintSpec`

```
<elementSpec ident="div" mode="change">
  <constraintSpec ident="attReq" scheme="schematron">
    <constraint>
      <s:report test="@n">
        L'élément div NE doit PAS avoir un
        attribut @n
      </s:report>
    </constraint>
  </constraintSpec>
</elementSpec>
```

# pattern

Pour combiner différentes `rule/assert` dans la même `constraint` et définir des variables, etc.

```
<constraintSpec ident="chapNb" scheme="schematron">
  <constraint>
    <s:pattern>
      <s:assert test="@n > 1">
        L'attribut @n doit être supérieur à 1
      </s:assert>
      <s:assert test="matches(@n, '^[1-9]\d*$')">
        L'attribut @n doit être un entier positif
      </s:assert>
    </s:pattern>
  </constraint>
</constraintSpec>
```

# Fonctions courantes

<code>count(node-set)</code>	Compte le nombre d'occurrences dans un ensemble de nœud
<code>sum(node-set)</code>	Somme des valeurs de type nombre dans un ensemble de nœud (qui ne contient <b>que des valeurs numériques</b> )
<code>true(), false(), not(bool)</code>	Vrai / Faux / Valeur inverse du booléen
<code>string(val), number(val)</code>	Pour changer le type d'une valeur
<code>concat(str, str)</code>	Concaténation de plusieurs chaînes de caractère
<code>start-with(str1, str2)</code>	Renvoie vrai si <code>str1</code> commence par <code>str2</code>
<code>contains(str1, str2)</code>	Renvoie vrai si <code>str1</code> contient <code>str2</code>
<code>string-length(str)</code>	Renvoie la longueur d'une chaîne de caractère
<code>normalize-space(str)</code>	Retire les espaces en début et fin de chaîne, ainsi que les doubles espaces
<code>last()</code>	Dernier nœud du document correspondant au prédicat
<code>match(str, regex)</code>	Renvoie vrai si <code>str</code> matche la regex (e.g. <code>'^[1-9]\d*\$'</code> )



# Exercice

**Rédiger une contrainte**

**<s:assert>**

# Consigne

- Reprendre l'ODD pour sonnetTEI.xml
- Rendre obligatoire la présence d'un ou plusieurs vers directement ou non dans un élément <lg> avec <s:assert> :
  - Trouver XPath pour les <l> descendants de <lg>
  - Trouver la fonction XPath pour compter le nombre de nœuds
  - Trouver comment tester que le nombre de nœuds correspond à "un ou plusieurs"
- Générer le schéma RelaxNG et l'associer au fichier TEI

# Ajouter du contexte

`<s:rule>` permet d'ajouter un contexte d'application à `<s:assert>`

`@context` permet de renseigner n'importe quel XPath, pas seulement relatif à l'élément concerné par `<elementSpec>`

```
<constraintSpec ident="subDiv" scheme="schematron">
  <constraint>
    <s:rule context="tei:div">
      <s:assert test="count(tei:div) != 1">
        Si elle contient des subdivisions,
        une div doit en contenir au moins deux
      </s:assert>
    </s:rule>
  </constraint>
</constraintSpec>
```

# Contraindre l'existence

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">
        Le début et la fin d'un lemme doivent
        être identifiés
      </s:assert>
    </s:rule>
  </constraint>
</constraintSpec>
```

# Exercice



**Rédiger une contrainte  
avec `<s:rule>`**

# Consigne

- Reprendre l'ODD pour sonnetTEI.xml
- Un `<lg type="sonnet">` doit commencer par un titre :
  - Trouver quel est le `@context`
  - Trouver quel caractère XPath permet de matcher n'importe quel élément
  - Trouver comment matcher un élément en première position
- Générer le schéma RelaxNG et l'associer au fichier TEI

# Contraindre le contenu

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

```
<constraintSpec ident="fromVal" scheme="schematron">  
  <constraint>  
    <s:rule context="tei:app[@from]">  
      <s:assert test="matches(@from, '^#w\d+$')">  
        L'attribut @from doit contenir une  
        valeur qui commence par #w et finit  
        par un nombre  
      </s:assert>  
    </s:rule>  
  </constraint>  
</constraintSpec>
```

# Déclarer une variable

Avec `s:let` et `@value` pour en définir la valeur

```
<s:rule context="tei:TEI//tei:body//tei:placeName">
  <s:let name="ref" value="substring-after(@ref, '#')"/>
  <s:assert test="@ref and
    $ref = //tei:settingDesc//tei:place/@xml:id">
    la valeur @ref du placeName doit être renseignée
    et correspondre à une place déclarée dans le
    settingDesc
  </s:assert>
</s:rule>
```



# **Tester les règles de validation**

**Pas d'erreur dans l'encodage  $\neq$  règle fonctionnelle**


⇒ Il faut essayer de briser la règle pour voir si une erreur apparaît

# Exercice

**Rédiger des règles  
Schematron**



# Consigne

- Reprendre l'ODD pour sonnetTEI.xml
- Paramétrer les `<lg type="sonnet">` de telle sorte à ce qu'il contiennent deux quatrains et un sizain
- Écrire une règle schematron pour que les valeurs @n de `<l>` se suivent :  
`number(@n) = number(preceding-sibling::tei:l[1]/@n) + 1`  
 Attention, il y a un piège (`<s:rule>`)
- Générer le schéma RelaxNG et l'associer au fichier TEI