

# Humanités numériques : structuration des données et des documents textuels

E. ROUQUETTE

Cours 4 – 14 février 2024

Contraindre l'encodage : le rôle des modèles

**Modéliser** : encoder un texte de façon à montrer son fonctionnement (structure, sémantisme)

→ une certaine façon de lire un texte, de sélectionner (simplifier) l'information qu'il contient.

Exemple : édition diplomatique vs édition critique

Modéliser un texte implique de **faire des choix**

## Exemple de choix : le nom des balises

Comment appeler l'entité désignant un paragraphe ?

`<p>`, `<par>`, `<paragraphe>` ?

## Exemple de choix : la granularité

### Typier les noms propres ?

```
<nomPersonne>Charles-Louis-Joseph Destable</nomPersonne>
```

ou

```
<nomPropre type="personne">Charles-Louis-Joseph  
Destable</nomPropre>
```

## Exemple de choix : la granularité

### Décrire les composants du nom ?

`<nomPersonne>Charles-Louis-Joseph Destable</nomPersonne>`

ou

`<nomPersonne><prenom>Charles-Louis-Joseph</prenom>`

`<nom>Destable</nom></nomPersonne>`

## Exemple de choix : la granularité

Associer à chaque nom une référence ?

```
<nomPersonne>L'abbé Destable</nomPersonne>
```

ou

```
<nomPersonne id="Destable">Charles-Louis-Joseph  
Destable</nomPersonne>
```

```
<nomPersonne ref="#Destable">L'abbé  
Destable</nomPersonne>
```

```
<nomPersonne ref="#Destable" langue="latin">Carolus  
Ludovicus Josephus Destable</nomPersonne>
```

## Exemple de choix : la granularité

indiquer le formatage, la langue ?

```
<epigraphe>carolus ludovicus josephus destable,  
aetatis suae 30 david laureti pinxit romæ</epigraphe>
```

ou

```
<epigraphe casse="maj" langue="latin">carolus  
ludovicus josephus destable, aetatis suae 30 david  
laureti pinxit romæ</epigraphe>
```



## Exemple de choix : la place des éléments

### Place des entités les unes par rapports aux autres

```
<tampon>Imprimés Bibliothèque nationale de  
  ↪  France</tampon>  
<titre>...</titre>
```

ou

```
<titre>...</titre>  
<tampon>Imprimés Bibliothèque nationale de  
  ↪  France</tampon>
```

## Exemple de choix : la place des éléments

### Ordre des attributs

```
<epigraphe casse="maj" langue="latin">...</epigraphe>
```

ou

```
<epigraphe langue="latin" casse="maj">...</epigraphe>
```

## Exemple de choix : le niveau de structuration

### Faiblement structuré

<p>Dans la salle capitulaire des chanoines de l'insigne Église de Reims se trouve, richement encadré, peint par un grand maître sicilien, le portrait d'un prêtre, provenant de M. Tourneur-Lemontier, son neveu. L'abbé est en manchettes de dentelles, les cheveux roulés et poudrés, portant, suspendue par un large ruban rouge, une croix à quatre branches unies par des rayons, le tout en or.</p>

## Exemple de choix : le niveau de structuration

### Moyennement structuré

<p>Dans la salle capitulaire des chanoines de l'insigne Église de <nomPropre type="lieu">Reims </nomPropre> se trouve, richement encadré, peint par un grand maître sicilien, le por<cesure/>trait d'un prêtre, provenant de M. <nomPropre type="personne">Tourneur-Lemontier</nomPropre>, son neveu. L'abbé est en manchettes de dentelles, les cheveux roulés et poudrés, portant, suspendue par un large ruban rouge, une croix à quatre branches unies par des rayons, le tout en or.</p>

## Exemple de choix : le niveau de structuration

### Fortement structuré

`<p n="001">`Dans la salle capitulaire des chanoines de l'insigne Église de `<nomPropre type="lieu">`Reims `</nomPropre>` se`<changementLigne/>` trouve, richement encadré, peint par un grand maître sicilien, le por`<changementLigne type="cesure"/>`trait d'un prêtre, provenant de M. `<nomPropre type="personne">`Tourneur-Lemontier`</nomPropre>`, son neveu. L'abbé`<changementLigne/>` est en manchettes de dentelles, les cheveux roulés et poudrés, portant,`<changementLigne/>` suspendue par un large ruban rouge, une croix à quatre branches unies`<changementLigne/>` par des rayons, le tout en or.`</p>`

# Exemple de choix : le niveau de structuration

## Très fortement structuré...

```
<p n="001"><mot n="00001">Dans</mot> <mot  
  ↳ n="00002">la</mot> <mot n="00003">salle</mot>  
  ↳ <mot n="00004">capitulaire</mot> <mot  
  ↳ n="00005">des</mot> <mot  
  ↳ n="00006">chanoines</mot> <mot  
  ↳ n="00007">de</mot> <mot n="00008">l'</mot> <mot  
  ↳ n="00009">insigne</mot> <mot  
  ↳ n="00010">Église</mot> <mot n="00011">de</mot>  
  ↳ <mot n="00012"><nomPropre  
  ↳ type="lieu">Reims</nomPropre></mot> <mot  
  ↳ n="00013">se</mot><changementLigne n="001"/>  
  ....</p>
```

# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

**Définir** Créer un **format adapté** à l'information que l'on souhaite stocker ; il faut donc **penser en amont** les données que l'on veut encoder.



# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

- Définir** Créer un **format adapté** à l'information que l'on souhaite stocker ; il faut donc **penser en amont** les données que l'on veut encoder.
- Guider** Plus grande facilité à **ajouter de nouvelles données** si celles-ci suivent un modèle : autocomplétion.

# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

- Définir** Créer un **format adapté** à l'information que l'on souhaite stocker ; il faut donc **penser en amont** les données que l'on veut encoder.
- Guider** Plus grande facilité à **ajouter de nouvelles données** si celles-ci suivent un modèle : autocomplétion.
- Exploiter** **Automatisation** du traitement des données (lorsque utilisation du document via des processeurs) possible seulement si elles suivent le même format.

# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

- Définir** Créer un **format adapté** à l'information que l'on souhaite stocker ; il faut donc **penser en amont** les données que l'on veut encoder.
- Guider** Plus grande facilité à **ajouter de nouvelles données** si celles-ci suivent un modèle : autocomplétion.
- Exploiter** **Automatisation** du traitement des données (lorsque utilisation du document via des processeurs) possible seulement si elles suivent le même format.
- Collaborer** Existence d'un modèle indispensable dans un projet où **différentes personnes** encodent dans un but commun.

# Pourquoi contraindre le XML : à quoi servent les modèles

Le but des modèles est d'**uniformiser** la façon dont on va modéliser (encoder) un texte ou un ensemble de textes donnés, afin de :

**Définir** Créer un **format adapté** à l'information que l'on souhaite stocker ; il faut donc **penser en amont** les données que l'on veut encoder.

**Guider** Plus grande facilité à **ajouter de nouvelles données** si celles-ci suivent un modèle : autocomplétion.

**Exploiter** **Automatisation** du traitement des données (lorsque utilisation du document via des processeurs) possible seulement si elles suivent le même format.

**Collaborer** Existence d'un modèle indispensable dans un projet où **différentes personnes** encodent dans un but commun.

**Documenter** **Documenter** le travail (les choix d'encodage) : important d'un point de vue scientifique mais aussi utile pour l'échange des données.

## Rappel : Conformité et validité

**Conformité** respect de la syntaxe XML (pas de chevauchement, ouverture/fermeture des balises, élément racine)

## Rappel : Conformité et validité

**Conformité** respect de la syntaxe XML (pas de chevauchement, ouverture/fermeture des balises, élément racine)

**Validité** conformité + respect de la structure définie par un modèle (nom des balises, des attributs, ordre des éléments, références résolues)

## Rappel : Conformité et validité

**Conformité** respect de la syntaxe XML (pas de chevauchement, ouverture/fermeture des balises, élément racine)

**Validité** conformité + respect de la structure définie par un modèle (nom des balises, des attributs, ordre des éléments, références résolues)

→ un document peut être **conforme** sans être **valide**

→ tout ce qui n'est **pas spécifié** (présenté dans le document de spécification, le modèle) est **interdit** (non valide)

# Les spécifications XML (modèles/schémas)

**Spécification** Ensemble des règles à suivre

**Schéma** Description de la structure que doit respecter un document qui lui fait référence. On parle aussi de *grammaire*, qui vient en complément de la *syntaxe* XML.

→ Un schéma **spécifie** un ensemble de noms d'**éléments**, les noms et le type de données de tous les **attributs** qui leur sont associés, et les **règles** relatives aux contextes dans lesquels ils peuvent apparaître

« Les éléments autorisés sont : **<document>**, **<p>**, **<titre>** et **<nomPersonne>**. **<document>** doit contenir : **<p>**, peut contenir : **<titre>**. **<p>** peut contenir **<nomPersonne>** et du texte. **<p>** peut avoir un attribut **@n**, qui doit avoir comme valeur un contenu alphanumérique. »

Exemple d'expression d'un morceau de schéma XML en langage naturel



# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément **<p>**

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément **<p>**
- ▶ l'existence d'un attribut **@n**

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément **<p>**
- ▶ l'existence d'un attribut **@n**
- ▶ l'existence d'une valeur **"3"**

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément **<p>**
- ▶ l'existence d'un attribut **@n**
- ▶ l'existence d'une valeur **"3"**

Et leur **arborescence** :

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément `<p>`
- ▶ l'existence d'un attribut `@n`
- ▶ l'existence d'une valeur `"3"`

Et leur **arborescence** :

- ▶ la possibilité pour un élément ou un attribut de contenir un autre élément, un attribut ou une valeur (option) `<p>` ou `<p n="x">`

# Règles d'un schéma

On peut spécifier la **présence** d'éléments :

- ▶ l'existence d'un élément `<p>`
- ▶ l'existence d'un attribut `@n`
- ▶ l'existence d'une valeur `"3"`

Et leur **arborescence** :

- ▶ la possibilité pour un élément ou un attribut de contenir un autre élément, un attribut ou une valeur (option) `<p>` ou `<p n="x">`
- ▶ l'obligation de les contenir (obligation) `<p n="x">`

# Règles d'un schéma

On peut spécifier les **occurrences** des éléments :



# Règles d'un schéma

On peut spécifier les **occurrences** des éléments :

- ▶ la possibilité de choisir entre différents enfants/attributs/valeurs, de manière optionnelle (choix optionnel) `<p n="x">` ou `<p type="titre">` ou `<p>`

# Règles d'un schéma

On peut spécifier les **occurrences** des éléments :

- ▶ la possibilité de choisir entre différents enfants/attributs/valeurs, de manière optionnelle (choix optionnel) `<p n="x">` ou `<p type="titre">` ou `<p>`
- ▶ le choix entre plusieurs éléments enfants/attributs/valeurs (choix obligatoire) `<p n="x">` ou `<p type="titre">`

# Règles d'un schéma

On peut spécifier les **occurrences** des éléments :

- ▶ la possibilité de choisir entre différents enfants/attributs/valeurs, de manière optionnelle (choix optionnel) `<p n="x">` ou `<p type="titre">` ou `<p>`
- ▶ le choix entre plusieurs éléments enfants/attributs/valeurs (choix obligatoire) `<p n="x">` ou `<p type="titre">`
- ▶ l'ordre des éléments `<titre>` `<auteur>` `<date>`, et pas `<auteur>` `<date>` `<titre>`

# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

- ▶ Un modèle XML doit répondre à un besoin donné et lui être adapté

# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

- ▶ Un modèle XML doit répondre à un besoin donné et lui être adapté
- ▶ Un modèle XML doit viser à l'absence d'ambiguïté et à être explicite (pour la machine et la personne)

# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

- ▶ Un modèle XML doit répondre à un besoin donné et lui être adapté
- ▶ Un modèle XML doit viser à l'absence d'ambiguïté et à être explicite (pour la machine et la personne)
- ▶ Un modèle XML tend à être essentiellement sémantique

# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

- ▶ Un modèle XML doit répondre à un besoin donné et lui être adapté
- ▶ Un modèle XML doit viser à l'absence d'ambiguïté et à être explicite (pour la machine et la personne)
- ▶ Un modèle XML tend à être essentiellement sémantique
- ▶ Un modèle XML doit être documenté



# Un bon modèle

Un modèle est toujours une **simplification de la complexité** du donné. Cette simplification en **facilite l'analyse** en sélectionnant des faits et en les formalisant.

- ▶ Un modèle XML doit répondre à un besoin donné et lui être adapté
- ▶ Un modèle XML doit viser à l'absence d'ambiguïté et à être explicite (pour la machine et la personne)
- ▶ Un modèle XML tend à être essentiellement sémantique
- ▶ Un modèle XML doit être documenté
- ▶ Un modèle XML doit pouvoir garder la capacité d'évoluer

# Les types de modèles

► **DTD** (*Document Type Definition*) (format .dtd).

**avantages** Assez simple ; efficace dans la spécification des langages de description documentaire (ex : [www.docbook.org](http://www.docbook.org))

**inconvénients** Syntaxe non XML (héritée de SGML), alors que XML est un métalangage... ; assez limité

# Les types de modèles

## ► **Schémas XML (W3C Schema)** (format .xsd)

**avantages** Sont en XML ; norme du W3C ; permet un typage très fin des données, l'héritage des propriétés,...

**inconvénient** l'emboîtement des balises XML peut être lourde à lire

## ► **Schémas Relax NG** (REgular LAnguage for Xml Next Generation) en syntaxe compacte (format .rnc) ou en syntaxe XML (format .rng)

**avantages** plus simple, souple et très utilisé. La TEI lui est historiquement liée

## ► **Schematron** en XML, basé sur XPath (format .sch)

**avantages** Permet des tests et des contraintes complexes

**inconvénient** Tout ce qui n'est pas défini est considéré comme valide

# Les types de modèles

- ▶ une spécification peut être rédigée en un seul langage ou en plusieurs
- ▶ Des conversions (via XSLT) d'un langage de schéma en un autre sont possibles.

# Comprendre une DTD

## Les opérateurs

Opérateur	Description
*	0 , 1 ou plusieurs
?	0 ou 1
+	1 ou plusieurs (au moins 1)
<i>rien</i>	exactement 1
	ou bien
,	et

**Déclarer un élément**

## Déclarer un élément

```
<!ELEMENT element>
```

## Déclarer un élément

`<!ELEMENT element>`

- ▶ Si l'élément contient uniquement du texte :

`<!ELEMENT element (#PCDATA)>`



## Déclarer un élément

`<!ELEMENT element>`

- ▶ Si l'élément contient uniquement du texte :

`<!ELEMENT element (#PCDATA)>`

- ▶ Si l'élément est vide :

`<!ELEMENT element EMPTY>`

## Déclarer des éléments enfants

## Déclarer des éléments enfants

- ▶ des éléments enfants selon un ordre obligatoire :

```
<!ELEMENT element (enfant1,enfant2,enfant3)>
```

## Déclarer des éléments enfants

- ▶ des éléments enfants selon un ordre obligatoire :  
`<!ELEMENT element (enfant1,enfant2,enfant3)>`
- ▶ Déclarer des éléments enfants autorisés en choix :  
`<!ELEMENT element (enfant1|enfant2|enfant3)>`

## Déclarer des éléments enfants

- ▶ des éléments enfants selon un ordre obligatoire :

```
<!ELEMENT element (enfant1,enfant2,enfant3)>
```

- ▶ Déclarer des éléments enfants autorisés en choix :

```
<!ELEMENT element (enfant1|enfant2|enfant3)>
```

- ▶ Précisions sur les éléments enfants autorisés :

```
<!ELEMENT element (enfant1,enfant2*,enfant3+,enfant4?)>
```

## Déclarer des éléments enfants

- ▶ des éléments enfants selon un ordre obligatoire :

```
<!ELEMENT element (enfant1,enfant2,enfant3)>
```

- ▶ Déclarer des éléments enfants autorisés en choix :

```
<!ELEMENT element (enfant1|enfant2|enfant3)>
```

- ▶ Précisions sur les éléments enfants autorisés :

```
<!ELEMENT element (enfant1,enfant2*,enfant3+,enfant4?)>
```

- ▶ Déclarer des éléments mixtes selon un ordre obligatoire :

```
<!ELEMENT element (#PCDATA,enfant1)>
```

## Déclarer des éléments enfants

- ▶ des éléments enfants selon un ordre obligatoire :

```
<!ELEMENT element (enfant1,enfant2,enfant3)>
```

- ▶ Déclarer des éléments enfants autorisés en choix :

```
<!ELEMENT element (enfant1|enfant2|enfant3)>
```

- ▶ Précisions sur les éléments enfants autorisés :

```
<!ELEMENT element (enfant1,enfant2*,enfant3+,enfant4?)>
```

- ▶ Déclarer des éléments mixtes selon un ordre obligatoire :

```
<!ELEMENT element (#PCDATA,enfant1)>
```

- ▶ Déclarer des éléments mixtes, contenu au choix :

```
<!ELEMENT element (#PCDATA|enfant1)*>
```

## Exercice : interpréter une DTD (1)

Que veut dire :

```
<!ELEMENT metadonnees
```

```
  ↪ (auteur,titre,soustitre*,detail+,date)>
```

```
<!ELEMENT poeme (titre,strophe*,vers+)>
```

```
<!ELEMENT strophe (vers+)>
```

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



## Déclarer des attributs

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

Exemple de déclaration de l'attribut `@when` sur l'élément `<date>` :

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

Exemple de déclaration de l'attribut `@when` sur l'élément `<date>` :

- ▶ attribut obligatoire, comprenant n'importe quelle séquence de caractères

```
<!ATTLIST date when CDATA #REQUIRED>
```

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

Exemple de déclaration de l'attribut `@when` sur l'élément `<date>` :

- ▶ attribut obligatoire, comprenant n'importe quelle séquence de caractères

```
<!ATTLIST date when CDATA #REQUIRED>
```

- ▶ attribut optionnel :

```
<!ATTLIST date when CDATA #IMPLIED>
```

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

Exemple de déclaration de l'attribut @when sur l'élément `<date>` :

- ▶ attribut obligatoire, comprenant n'importe quelle séquence de caractères

```
<!ATTLIST date when CDATA #REQUIRED>
```

- ▶ attribut optionnel :

```
<!ATTLIST date when CDATA #IMPLIED>
```

- ▶ Ensemble de valeurs possibles pour l'attribut :

```
<!ATTLIST date when (avJC|apJC) #REQUIRED>
```

## Déclarer des attributs

Syntaxe : `<!ATTLIST element attribut donnees mode>`

Exemple de déclaration de l'attribut `@when` sur l'élément `<date>` :

- ▶ attribut obligatoire, comprenant n'importe quelle séquence de caractères

```
<!ATTLIST date when CDATA #REQUIRED>
```

- ▶ attribut optionnel :

```
<!ATTLIST date when CDATA #IMPLIED>
```

- ▶ Ensemble de valeurs possibles pour l'attribut :

```
<!ATTLIST date when (avJC|apJC) #REQUIRED>
```

Ce qui donnera par exemple comme encodage valide :

```
<date when="avJC">300</date> ou :
```

```
<date when="apJC">300</date>
```

## Exercice : interpréter une DTD (2)

Que veut dire :

```
<!ELEMENT poeme (strophe*)>
```

```
<!ATTLIST strophe type (quatrain|tercet|sizain) #REQUIRED>
```



# Associer une DTD à un document

Les spécifications d'une DTD peuvent être rédigées :

- ▶ dans le document lui-même (DTD interne)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?  
<!DOCTYPE personne [  
  <!--début de la DTD interne -->  
  <!ELEMENT personne (prenom, nom)>  
  <!ELEMENT prenom (#PCDATA)>  
  <!ELEMENT nom (#PCDATA)>  
  <!--fin de la DTD interne -->  
>
```

- ▶ dans un fichier auquel le document va ensuite faire référence par une instruction (DTD externe), entre le prologue et l'élément racine :

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE personne SYSTEM "modele.dtd">  
<!--début du document-->
```

# Associer un schéma à un document


Les spécifications des schémas (non DTD) doivent être rédigées dans un fichier auquel le document va ensuite faire référence au niveau de l'élément racine :

```
<?xml version="1.0"?>
```

```
<reference
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:noNamespaceSchemaLocation="monSchema.xsd">
```

# Exercice : générer et tester un schéma dans oXygen

- ▶ À partir du poème *L'Adieu* encodé la semaine dernière, générer un schéma XML avec oXygen :  
document - schéma - générer/convertir le schéma -  
WRC XML schema
- ▶ associer le schéma au document : icône épingle rouge . Il est bien sûr valide...
- ▶ Créer un nouveau document, y mettre le texte *L'automne* d'Apollinaire
- ▶ Mettre la même balise racine pour encoder le poème *L'Automne*, puis associer le schéma au document
- ▶ L'employer pour encoder le poème *L'Automne*.
- ▶ Autocomplétion des balises et attributs
- ▶ Un certain ordre attendu

## Exercice : générer et tester un schéma dans oXygen

- ▶ Dans le document contenant Automne, enlever l'appel du schéma
- ▶ Ajouter dans le poème des balises `<strophe>...</strophe>`
- ▶ générer à nouveau le schéma ; l'associer au document : il est valide
- ▶ l'associer au poème *L'Adieu*

→ Problème : notre encodage de *L'Adieu* n'est plus valide :

- ▶ car « L'Adieu n'est pas une valeur valide pour NCName »
- ▶ car il n'a pas d'élément `<strophe>`

# Exercice : modifier et tester un schéma dans oXygen

- ▶ Ouvrir le schéma

document - schéma - ouvrir le schéma associé et trouver la définition de l'élément `<titre>` :

13 `<xs:element name="titre" type="xs:NCName">`

- ▶ Remplacer « NCName » par « string », comme pour l'élément `<auteur>`

# Exercice : modifier et tester un schéma dans oXygen

- ▶ Ouvrir le schéma

document - schéma - ouvrir le schéma associé et trouver la définition de l'élément `<titre>` :

13 `<xs:element name="titre" type="xs:NCName">`

- ▶ Remplacer « NCName » par « string », comme pour l'élément `<auteur>`

## explication :

- ▶ NCName : « non-colonized name ». Un nom qui ne peut pas contenir les signes `$ % & + ; ) (` ni espace blanc
- ▶ string : une chaîne de caractères

## Exercice : modifier et tester un schéma dans oXygen

- Trouver la définition de l'élément `<poeme>` :

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:sequence>
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17     </xs:sequence>
18   </xs:complexType>
19 </xs:element>
```

# Exercice : modifier et tester un schéma dans oXygen

- Trouver la définition de l'élément `<poeme>` :

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:sequence>
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17     </xs:sequence>
18   </xs:complexType>
19 </xs:element>
```

**explication :**



## Exercice : modifier et tester un schéma dans oXygen

- Trouver la définition de l'élément `<poeme>` :

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:sequence>
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17     </xs:sequence>
18   </xs:complexType>
19 </xs:element>
```

### explication :

- `<xs:complexType>` : élément constitué d'autres éléments

## Exercice : modifier et tester un schéma dans oXygen

- Trouver la définition de l'élément `<poeme>` :

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:sequence>
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17     </xs:sequence>
18   </xs:complexType>
19 </xs:element>
```

### explication :

- `<xs:complexType>` : élément constitué d'autres éléments
- `<xs:sequence>` : indique que les éléments enfants sont obligatoires

## Exercice : modifier et tester un schéma dans oXygen

→ utiliser à la place `<xs:choice>` : seulement un des sous-éléments peut apparaître, au choix.

## Exercice : modifier et tester un schéma dans oXygen

→ utiliser à la place `<xs:choice>` : seulement un des sous-éléments peut apparaître, au choix.

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:choice maxOccurs="unbounded">
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17       <xs:element maxOccurs="unbounded" ref="vers"/>
18     </xs:choice>
19   </xs:complexType>
20 </element>
```

## Exercice : modifier et tester un schéma dans oXygen

→ utiliser à la place `<xs:choice>` : seulement un des sous-éléments peut apparaître, au choix.

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:choice maxOccurs="unbounded">
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17       <xs:element maxOccurs="unbounded" ref="vers"/>
18     </xs:choice>
19   </xs:complexType>
20 </element>
```

- ▶ vérifier la validité de nos deux documents avec ce nouveau schéma

## Exercice : modifier et tester un schéma dans oXygen

→ utiliser à la place `<xs:choice>` : seulement un des sous-éléments peut apparaître, au choix.

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:choice maxOccurs="unbounded">
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17       <xs:element maxOccurs="unbounded" ref="vers"/>
18     </xs:choice>
19   </xs:complexType>
20 </element>
```

- ▶ vérifier la validité de nos deux documents avec ce nouveau schéma
- ▶ observer dans oXygen l'arbre du schéma (cliquer sur « Design »)

## Exercice : modifier et tester un schéma dans oXygen

→ utiliser à la place `<xs:choice>` : seulement un des sous-éléments peut apparaître, au choix.

```
13 <xs:element name="poeme">
14   <xs:complexType>
15     <xs:choice maxOccurs="unbounded">
16       <xs:element maxOccurs="unbounded" ref="strophe"/>
17       <xs:element maxOccurs="unbounded" ref="vers"/>
18     </xs:choice>
19   </xs:complexType>
20 </element>
```

- ▶ vérifier la validité de nos deux documents avec ce nouveau schéma
- ▶ observer dans oXygen l'arbre du schéma (cliquer sur « Design »)

Pour aller plus loin : <https://fabien-torre.fr/Enseignement/Cours/XML/xmlschema.php>

- ▶ RelaxNG (REgular LAnguage for XML Next Generation) est un standard du consortium OASIS (Organization for the Advancement of Structured Information Standards)
- ▶ Il a deux syntaxes : une syntaxe XML (alternative à W3C Schema) et une syntaxe compacte (alternative aux DTD).
- ▶ Utilisé notamment dans les formats OpenDocument, et **dans la TEI**



# Les schémas RelaxNG

Exemple en syntaxe XML (format .rng) :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <grammar xmlns="http://relaxng.org/ns/structure/1.0">
3    <start>
4      <element name="document">
5        <attribute name="type"/>
6        <oneOrMore>
7          <element name="paragraphe">
8            <text/>
9          </element>
10         </oneOrMore>
11       </element>
12     </start>
13 </grammar>
```

Source : <https://stph.scenari-community.org/doc/sch.pdf> p.7

# Les schémas RelaxNG

En syntaxe compacte (format .rnc) :

```
1 start = element document {  
2   attribute type {text},  
3   element paragraphe {text}+  
4 }
```

Source : <https://stph.scenari-community.org/doc/sch.pdf> p.7

# La TEI - Introduction : qu'est-ce que la TEI



# La TEI est un langage XML

- ▶ Text Encoding Initiative

# La TEI est un langage XML

- ▶ Text Encoding Initiative
- ▶ application d'un schéma XML

# La TEI est un langage XML

- ▶ Text Encoding Initiative
- ▶ application d'un schéma XML
- ▶ fournit des recommandations pour la création et la gestion de données, surtout textuelles

# La TEI est un langage XML

- ▶ Text Encoding Initiative
- ▶ application d'un schéma XML
- ▶ fournit des recommandations pour la création et la gestion de données, surtout textuelles
- ▶ naissance à la fin des années 1980 ; version actuelle, P5, depuis 2007

# La TEI est un langage XML

- ▶ Text Encoding Initiative
- ▶ application d'un schéma XML
- ▶ fournit des recommandations pour la création et la gestion de données, surtout textuelles
- ▶ naissance à la fin des années 1980 ; version actuelle, P5, depuis 2007

→ La TEI est donc un **langage XML** qui possède une définition précise de ses éléments, attributs, valeurs et de leurs agencements possibles.



# La TEI est un langage XML

Document XML : balises libres

```
<citation type="inscription"><vers>Le Jardin, le bon ton, l'  
<vers> Peut être anglais, français, chinois</vers>  
<vers>Mais les eaux les prés et les bois</vers>  
<vers>La nature et le paysage</vers>  
</citation>
```

Document XML/TEI : balises prédéfinies

```
<quote type="inscription"><l>Le Jardin, le bon ton, l'usage  
<l> Peut être anglais, français, chinois</l>  
<l>Mais les eaux les prés et les bois</l>  
<l>La nature et le paysage</l>  
</quote>
```

# La TEI est un consortium

- ▶ un consortium qui maintient et fait évoluer le langage
- ▶ existe depuis 1994
- ▶ constitué de chercheurs et chercheuses et de professionnels et professionnelles de l'information
- ▶ large communauté et nombreuses activités de réflexion (groupes de réflexion, liste de diffusion, conférence annuelle, etc.)

## Exemple d'éditions en XML-TEI

Liste des projets utilisant la TEI :

<https://tei-c.org/activities/projects/>

Quelques exemples :

Édition des capitulaires carolingiens : [https:](https://capitularia.uni-koeln.de/en/mss/paris-bn-lat-2718/)

[//capitularia.uni-koeln.de/en/mss/paris-bn-lat-2718/](https://capitularia.uni-koeln.de/en/mss/paris-bn-lat-2718/)

Base de Français Médiéval

<https://txm-bfm.huma-num.fr/txm/>

Les manuscrits de Stendhal <http://stendhal.demarre-shs.fr/>

# Utiliser la TEI

Pour utiliser un set de balises TEI, il faut déclarer **l'espace de nom TEI** (namespace) dans l'élément racine grâce à une **adresse URI**

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
```

Le logiciel oXygen le fait automatiquement lorsque l'on crée un document TEI : Nouveau - modèle du framework - TEI P5 - All

# Petit détour par les espaces de noms (*namespace*)

## Définitions

**Espace de nom (*namespace*)** Un espace de nom indique le vocabulaire XML dont sont issus les éléments et attributs utilisés dans un document XML.

**URI** *Uniform Resource Identifier*. Identifiant qui permet de désigner sans ambiguïté un document externe (ou plus généralement une ressource), par exemple des DTD.

Ressource sur les espaces de noms : <https://www.irif.fr/~carton/Enseignement/XML/Cours/support.html#chap.namespace>

## Petit détour par les espaces de noms (*namespace*)

- ▶ Un **espace de noms** est **identifié par un URI**, qui est la plupart du temps également une URL, permettant d'**accéder à la documentation** de l'espace de noms.
- ▶ **Combiner les espaces de noms** permet de ne pas avoir à redéfinir des balises déjà définies dans d'autres vocabulaires/langages XML
- ▶ On peut choisir **un espace de nom par défaut** et attribuer aux autres espaces de noms un **préfixe** servant à identifier leurs balises
- ▶ On peut appeler des espaces de nom seulement au niveau des éléments concernés ou de leur élément parent

# Petit détour par les espaces de noms (*namespace*)

## Appeler plusieurs espaces de noms :

`<elementRacine`

`xmlns="namespaceDesÉlémentsNonPréfixés"`

`xmlns:prefixe1="namespace1"`

`xmlns:prefixe2="namespace2" ...>`

## Exemple

```
1 <TEI xmlns="http://www.tei-c.org/ns/1.0"
2   xmlns:mml="http://www.w3.org/1998/Math/MathML"
3   xmlns:xml="http://www.w3.org/XML/1998/namespace">
```

1. Nous pourrions utiliser les éléments définis par la TEI sans les préfixer
2. Nous pourrions utiliser les éléments du vocabulaire MathML, en les préfixant avec `mml` : Exemple : `<mml:math>`.
3. Nous pourrions utiliser les éléments du vocabulaire défini par le W3C, en les préfixant avec `xml` : Exemple : `<xml:id>`.

# Petit détour par les espaces de noms (*namespace*)

## Appeler plusieurs espaces de noms :

`<elementRacine`

`xmlns="namespaceDesÉlémentsNonPréfixés"`

`xmlns:prefixe1="namespace1"`

`xmlns:prefixe2="namespace2" ...>`

## Exemple

```
1 <TEI xmlns="http://www.tei-c.org/ns/1.0"
2   xmlns:mml="http://www.w3.org/1998/Math/MathML"
3   xmlns:xml="http://www.w3.org/XML/1998/namespace">
```

**RQ** : dans la pratique, il n'est pas besoin d'appeler l'espace de nom du W3C, car les schémas TEI l'appellent déjà par défaut



## Exercice 1 : créer avec oXygen un document XML-TEI

Nouveau - modèle du framework - TEI P5 - All

Encoder de la façon suivante le premier vers de l'acte IV du *Cid* de Corneille, **en observant quelles sont les balises proposées par oXygen** avec la commande Ctl-E :

```
<sp>  
  <speaker>Don Diègue</speaker>  
  <l>Ô rage! ô désespoir! ô vieillesse ennemie!</l>  
</sp>
```

# Exercice 1 : créer avec oXygen un document XML-TEI

Nouveau - modèle du framework - TEI P5 - All

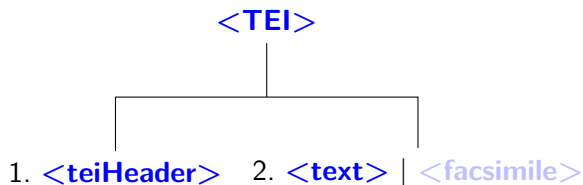
Encoder de la façon suivante le premier vers de l'acte IV du *Cid* de Corneille, **en observant quelles sont les balises proposées par oXygen** avec la commande Ctl-E :

```
<sp>  
  <speaker>Don Diègue</speaker>  
  <l>Ô rage! ô désespoir! ô vieillesse ennemie!</l>  
</sp>
```

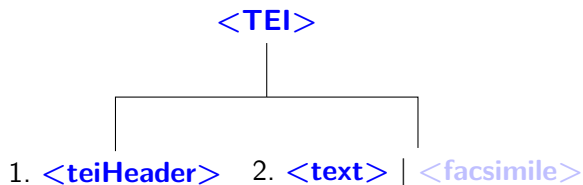
→ La balise **<speaker>** n'est pas proposée si l'on n'est pas dans un élément **<sp>**

## Les éléments obligatoires d'un document TEI

# La structure d'un document TEI

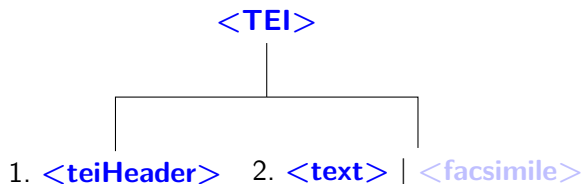


# La structure d'un document TEI



**<TEI>** l'élément racine

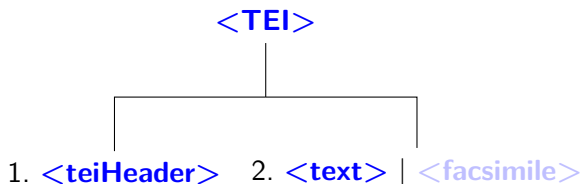
# La structure d'un document TEI



**<TEI>** l'élément racine

**<teiHeader>** l'élément contenant les métadonnées

# La structure d'un document TEI

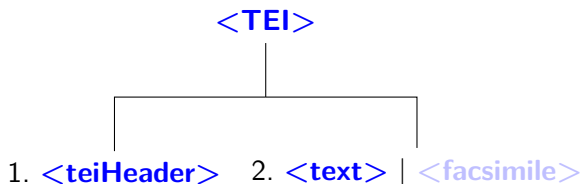


`<TEI>` l'élément racine

`<teiHeader>` l'élément contenant les métadonnées

`<text>` le texte encodé

# La structure d'un document TEI



**<TEI>** l'élément racine

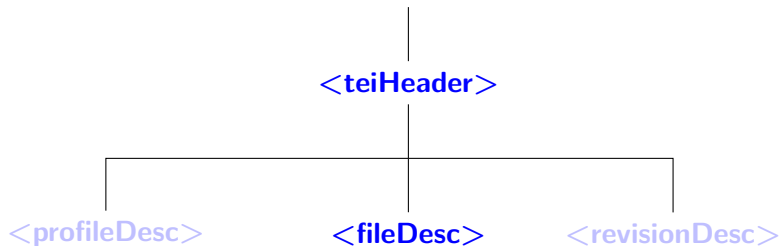
**<teiHeader>** l'élément contenant les métadonnées

**<text>** le texte encodé

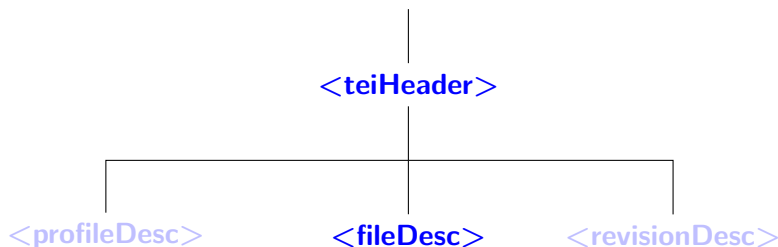
**<facsimile>** élément pouvant remplacer l'élément **<text>** pour des éditions en fac-similé (images plutôt que texte)



# Les métadonnées (<teiHeader>)

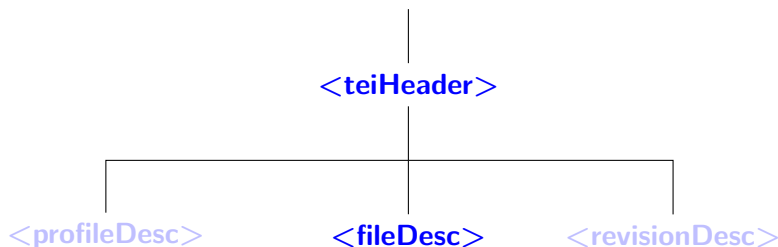


# Les métadonnées (<teiHeader>)



<fileDesc> « File description » : Description bibliographique du fichier électronique (**obligatoire**)

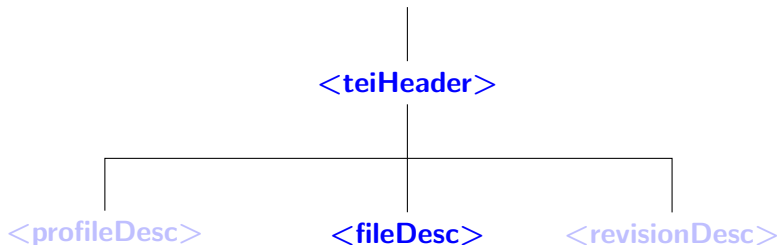
# Les métadonnées (<teiHeader>)



<fileDesc> « File description » : Description bibliographique du fichier électronique (**obligatoire**)

<profileDesc> « profile description » ; aspects non bibliographiques d'un texte (langue utilisée, participants, etc)

## Les métadonnées (<teiHeader>)

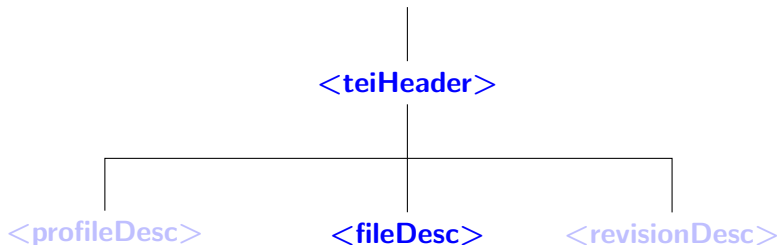


**<fileDesc>** « File description » : Description bibliographique du fichier électronique (**obligatoire**)

**<profileDesc>** « profile description » ; aspects non bibliographiques d'un texte (langue utilisée, participants, etc)

**<revisionDesc>** « Revision description » : Historique des révisions pour un fichier

## Les métadonnées (<teiHeader>)



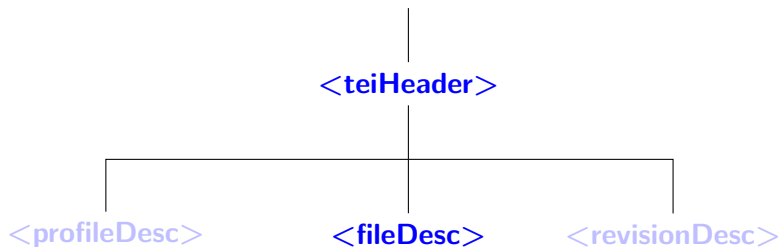
**<fileDesc>** « File description » : Description bibliographique du fichier électronique (**obligatoire**)

**<profileDesc>** « profile description » ; aspects non bibliographiques d'un texte (langue utilisée, participants, etc)

**<revisionDesc>** « Revision description » : Historique des révisions pour un fichier

... (d'autres éléments existent)

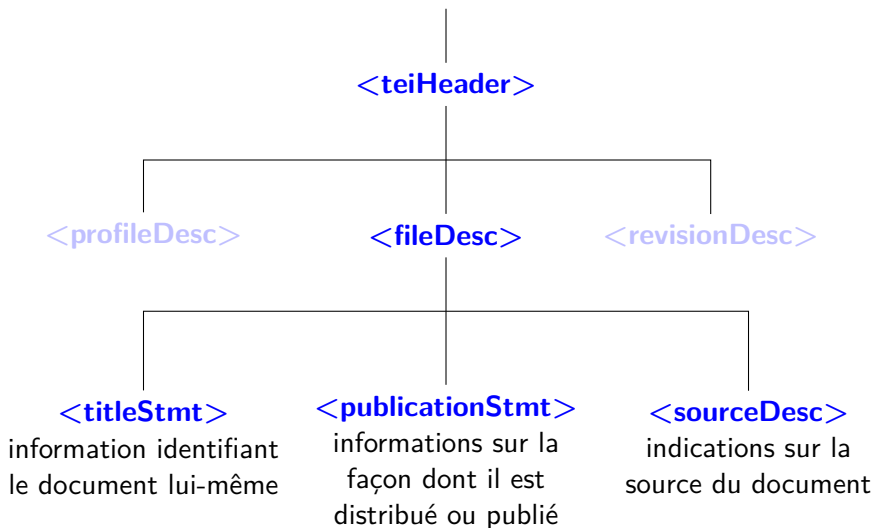
# Les métadonnées (<teiHeader>)



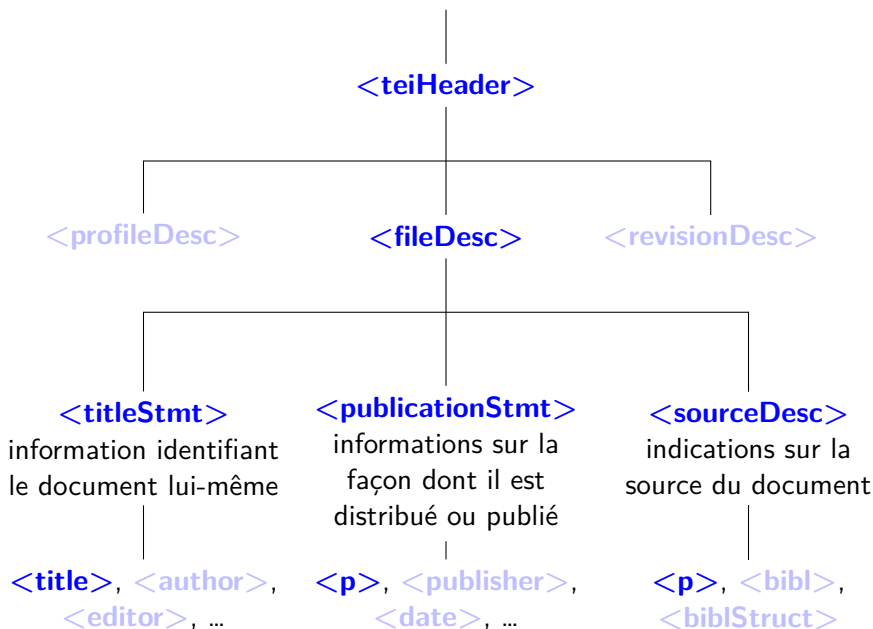
## Convention de nommage

- `_Stmt` « statement » : contenu structuré en sous-éléments
- `_Desc` « description » : description en prose, pouvant admettre des sous-éléments

# Les métadonnées (<teiHeader>)



# Les métadonnées (<teiHeader>)





## Les modules du *teiHeader*

Le `teiHeader` peut être étendu par différents modules optionnels en fonction de la nature du texte encodé :

- ▶ `msdescription` pour les manuscrits
- ▶ `textcrit` pour l'édition critique
- ▶ etc...

## Les métadonnées : exemples

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Le document TEI minimal</title>
    </titleStmt>
    <publicationStmt>
      <p>Distribué comme partie de TEI P5</p>
    </publicationStmt>
    <sourceDesc>
      <p>Aucune source : ce document est né
↪  numérique</p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

(source : TEI Guidelines, *The TEI Header and Its Components*)

# Les métadonnées : exemples

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Thomas Paine: Common sense, a machine-readable
↪ transcript</title>
      <respStmt>
        <resp>compiled by</resp>
        <name>Jon K Adams</name>
      </respStmt>
    </titleStmt>
    <publicationStmt>
      <distributor>Oxford Text Archive</distributor>
    </publicationStmt>
    <sourceDesc>
      <bibl>The complete writings of Thomas Paine, collected and
↪ edited by Phillip S. Foner (New York, Citadel Press,
↪ 1945)</bibl>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

(source : TEI Guidelines, *The TEI Header and Its Components*)

## Exercice 2

Encoder de façon simplifiée les métadonnées pour un document TEI dont les références seraient :

- ▶ Édition numérique du Capitulaire de Théodulf
- ▶ Dans le cadre du cours d'initiation à XML de l'EnC
- ▶ D'après l'édition de Peter Brommer (*MGH, Capitula episcoporum*, I)

Éléments à utiliser au sein des trois éléments obligatoires du `<fileDesc>` : `<title>`, `<author>`, `<p>`, `<bibl>`

# Solution

```
<teiHeader>
  <fileDesc>
    <titleStmt>
      <title>Édition numérique du Capitulaire de
↳ Théodulf</title>
    </titleStmt>
    <publicationStmt>
      <p>Non publié (cours d'initiation à XML de
↳ l'EnC)</p>
    </publicationStmt>
    <sourceDesc>
      <p>D'après l'édition de <bibl>Peter Brommer, MGH,
↳ Capitula episcoporum, I </bibl></p>
    </sourceDesc>
  </fileDesc>
</teiHeader>
```

## Pour la semaine prochaine

Lire l'introduction et les trois premiers chapitres de l'ouvrage de Lou Burnard sur la TEI