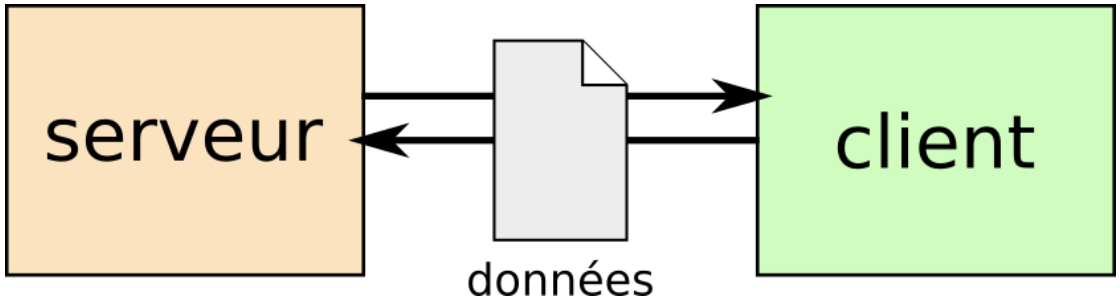
XML - Semaine 1

Pierre Nerzic

février-mars 2022

Remarque : le cours s'intitule « Data Internet » : étude des formats d'échange des données entre applications internet.



Plusieurs formats existent : XML, JSON, YAML, TOML, INI, CSV, etc. XML étant celui pour lequel il y a le plus d'outils et de concepts, il sera étudié principalement. Les autres seront évoqués à la fin.

Ce cours s'étend sur 8 semaines. Chaque semaine : 1h CM et 2h TP. Chaque TP est noté, il faut rendre le travail effectué en séance sur Moodle. Il n'y a pas de DS final.

Pierre Nerzic

Introduction

Pierre Nerzic

XML ?

XML = Extensible Markup Language

C'est un langage permettant de représenter et structurer des informations à l'aide de balises que chacun peut définir et employer comme il le veut.

```
texte ... <BALISE> ... texte </BALISE> ...
```

Le cours de cette semaine présente les concepts de base :

- Applications typiques,
- Historique,
- Structure d'un document XML.

Pierre Nerzic

Pourquoi ce cours ?

L'objectif est de connaître les principaux formats d'échanges de données entre logiciels : enregistrement de fichiers, communication entre serveur et clients, etc.

Les formats XML, JSON, YAML sont intéressants car ce sont des textes lisibles et structurés. Ils sont également accompagnés d'outils de vérification, transformation et interrogation. Ils sont directement liés à des structures de données dans la plupart des langages de programmation : un objet Java par exemple peut être lu ou écrit en une seule instruction dans un fichier JSON ou XML.

D'autres formats n'ont pas d'outils de traitement des fichiers.

Pourquoi ce cours, suite ?

Il faut noter que JavaScript semble prendre de plus en plus d'importance et il est fortement lié au format JSON.

Inversement Java semble traverser des turbulences avec des retards importants dans les nouvelles versions et des ruptures de compatibilité. Plusieurs API XML pour Java ne se compilent pas avec Java 9.

XML est le format le plus ancien, le plus polyvalent et le plus complet. C'est lui que nous étudierons principalement. Connaître XML permet d'utiliser les autres sans aucune difficulté.

Pierre Nerzic

Présentation rapide de XML

Pierre Nerzic

Exemple de fichier XML

Un fichier XML représente des informations structurées : 

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- itinéraire fictif -->
<itineraire>
    <etape distance="0km">départ</etape>
    <etape distance="13km">tourner à droite</etape>
    <etape distance="22km">arrivée</etape>
</itineraire>
```

Cet exemple modélise un itinéraire composé d'étapes.

XML permet de choisir la représentation des données sans aucune contrainte. On choisit les balises et les attributs comme on le souhaite.

Pierre Nerzic

Exemple de fichier XML, suite

Il faut seulement que ça soit homogène, régulier afin de pouvoir être exploité par un logiciel.

On peut faire ceci, mais bonjour les difficultés pour le traiter : 

```
<?xml version="1.0" encoding="utf-8"?>
<!-- itinéraire fictif -->
<itineraire>
    <etape distance="0km">départ</etape>
    <Etape><distance>13km</distance>tourner à droite</Etape>
    <etape distance="22km"><infos>arrivée</infos></etape>
</itineraire>
```


Pierre Nerzic

Applications de XML

Le format XML est au cœur de nombreux processus actuels :

- format d'enregistrement de nombreuses applications,
- échange de données entre serveurs et clients,
- outils et langages de programmation,
- bases de données XML natives.

Il faut comprendre que XML définit la syntaxe des documents, mais les applications définissent les balises, leur signification et ce qu'elles doivent contenir. Des API existent dans tous les langages pour lire et écrire des documents XML.

Voici une liste des applications un peu plus détaillée.

Pierre Nerzic

XML en tant que format de fichier

XML est la norme d'enregistrement de nombreux logiciels parmi lesquels on peut citer :

- Bureautique
  - LibreOffice : format OpenDocument
  - Publication de livres et documentations : DocBook
- Graphismes
  - Dessin vectoriel avec Inkscape : format SVG
  - Équations mathématiques : format MathML
- Programmation
  - Interfaces graphiques : Android, XUL (mais le futur WebExtensions est tourné vers JS et JSON)
  - Construction/compilation de projets : Ant, Maven
- Divers
  - Itinéraires GPS : format GPX

Pierre Nerzic

Bases de données XML

XML permet aussi de représenter des données complexes.

- Web sémantique : c'est un projet qui vise à faire en sorte que toutes les connaissances présentes plus ou moins explicitement dans les pages web puissent devenir accessibles par des mécanismes de recherche unifiés. Pour cela, il faudrait employer des marqueurs portant du sens en plus de marqueurs de mise en page, par exemple rajouter des balises indiquant que telle information est le prix unitaire d'un article.
  L'un des mécanismes du Web sémantique est une base de données appelée RDF. Elle peut être interrogée à l'aide d'un langage de requêtes appelé SparQL.
- Bases de données XML native : les données sont au format XML et les requêtes sont dans un langage (XQuery) permettant de réaliser l'équivalent de SQL.

Échange de données entre clients et serveur

XML est le format utilisé pour représenter des données volatiles de nombreux protocoles dont :

- Les flux RSS permettent de résumer les changements survenus sur un site Web. Par exemple, un site d'information émet des messages RSS indiquant les dernières nouvelles.
- Les protocoles XML-RPC et SOAP permettent d'exécuter des procédures à distance (*Remote Procedure Call*) : un client demande à un serveur d'exécuter un programme ou une requête de base de donnée puis attend les résultats. Le protocole gère l'encodage de la requête et des résultats, indépendamment des langages de programmation et des systèmes employés.
- Autre exemple : AJAX permet de mettre à jour dynamiquement les pages web sur le navigateur.

Historique

Pierre Nerzic

Origine de XML

XML est un cousin de HTML. Tous deux sont les successeurs de SGML, lui-même issu de GML. Ce dernier a été conçu par IBM pour dissocier l'apparence des documents textes de leur contenu, en particulier des documentations techniques. Il est facile de changer l'apparence (mise en page, polices, couleurs...) d'un tel document sans en ré-écrire une seule ligne. D'autres langages, comme LaTeXsont similaires : on écrit le texte sans se soucier de la mise en page. Cela permet également de le traduire dans d'autres langues.

Inversement, les logiciels *wysiwyg* comme Word mélangent mise en page et contenu. Même avec des styles, il reste difficile de remanier l'apparence d'un document sans devoir le ré-écrire partiellement.

XML permet de représenter beaucoup plus que des textes.

Chronologie

1969 : GML (IBM) GML est un langage d'écriture de documents techniques, défini par IBM, destiné à être traité par un logiciel propriétaire de mise en page appelé `SCRIPT/VS`.

1990 : SGML et HTML SGML est une norme libre très complexe dont HTML est un sous-ensemble très spécialisé. L'une des raisons qui ont conduit à les définir est d'améliorer la pérennité des documentations.

1998 : XML c'est une généralisation de SGML permettant de construire toutes sortes de documents.

Pierre Nerzic

Exemple de document GML

Les balises sont notées `:TAG.` ou `:TAG attr=valeur.` quand il y a des attributs. Les balises fermantes sont notées `:ETAG.` Cela ressemble beaucoup à la syntaxe `<TAG>` et `</TAG>`. L'une des plus utilisées est `:P.` pour créer un paragraphe.

```
:H1 id=exemple.Exemple de texte GML
:P.Certaines balises ressemblent un peu à celles de HTML
mais pas toutes. Voici du :HP2.texte en gras:EHP2. et un
exemple:FNREF refid=note1. :
:FN id=note1.Voici une note de bas de page.:EFN.
:XMP.
xmlstarlet c14n document.xml
:EXMP.
```


Pierre Nerzic

Structure d'un document XML

Pierre Nerzic

Arborescence d'éléments

Un document XML est composé de plusieurs parties :

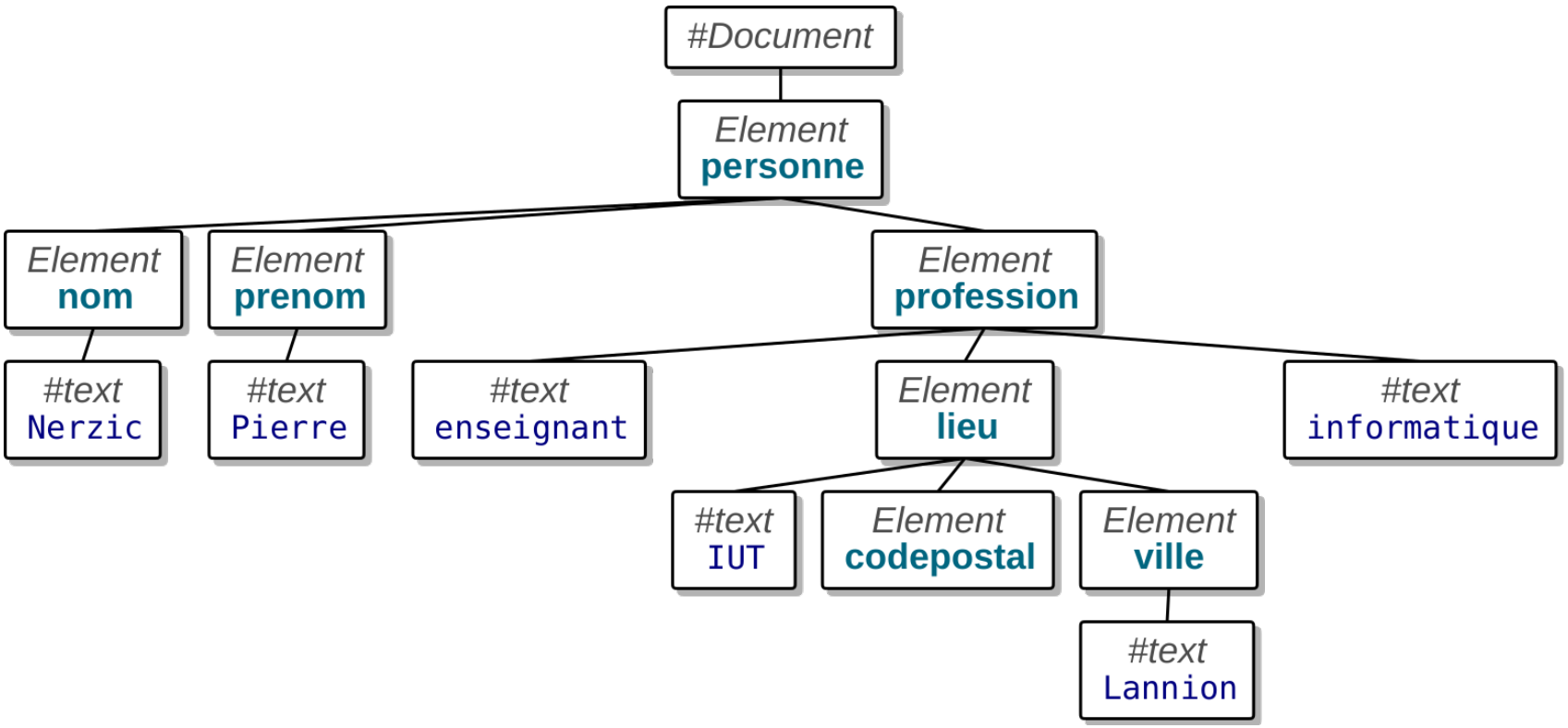
- Entête de document précisant la version et l'encodage,
- Des règles optionnelles permettant de vérifier si le document est valide (c'est l'objet des deux prochains cours),
- Un arbre d'*éléments* basé sur un élément appelé *racine*
  - Un *élément* possède un *nom*, des *attributs* et un *contenu*
  - Le contenu d'un élément peut être :
    - rien : élément vide noté `<nom/>` ou `<nom attributs.../>`
    - du texte
    - d'autres éléments (les éléments enfants).
  - Un élément non vide est délimité par une *balise ouvrante* et une *balise fermante*.
    - une balise ouvrante est notée `<nom attributs...>`
    - une balise fermante est notée `</nom>`

Exemple complet

Voici un document XML représentant une personne : 

```
<?xml version="1.0" encoding="utf-8"?>
<personne>
  <nom>Nerzic</nom>
  <prenom>Pierre</prenom>
  <profession>
    enseignant
    <lieu>
      IUT
      <codepostal/>
      <ville>Lannion</ville>
    </lieu>
    informatique
  </profession>
</personne>
```

Représentation graphique



Explications

Le document XML représente un arbre composé de plusieurs types de nœuds :

nœuds éléments ils sont associés aux balises `<element>`. Ce sont des nœuds qui peuvent avoir des enfants en dessous.

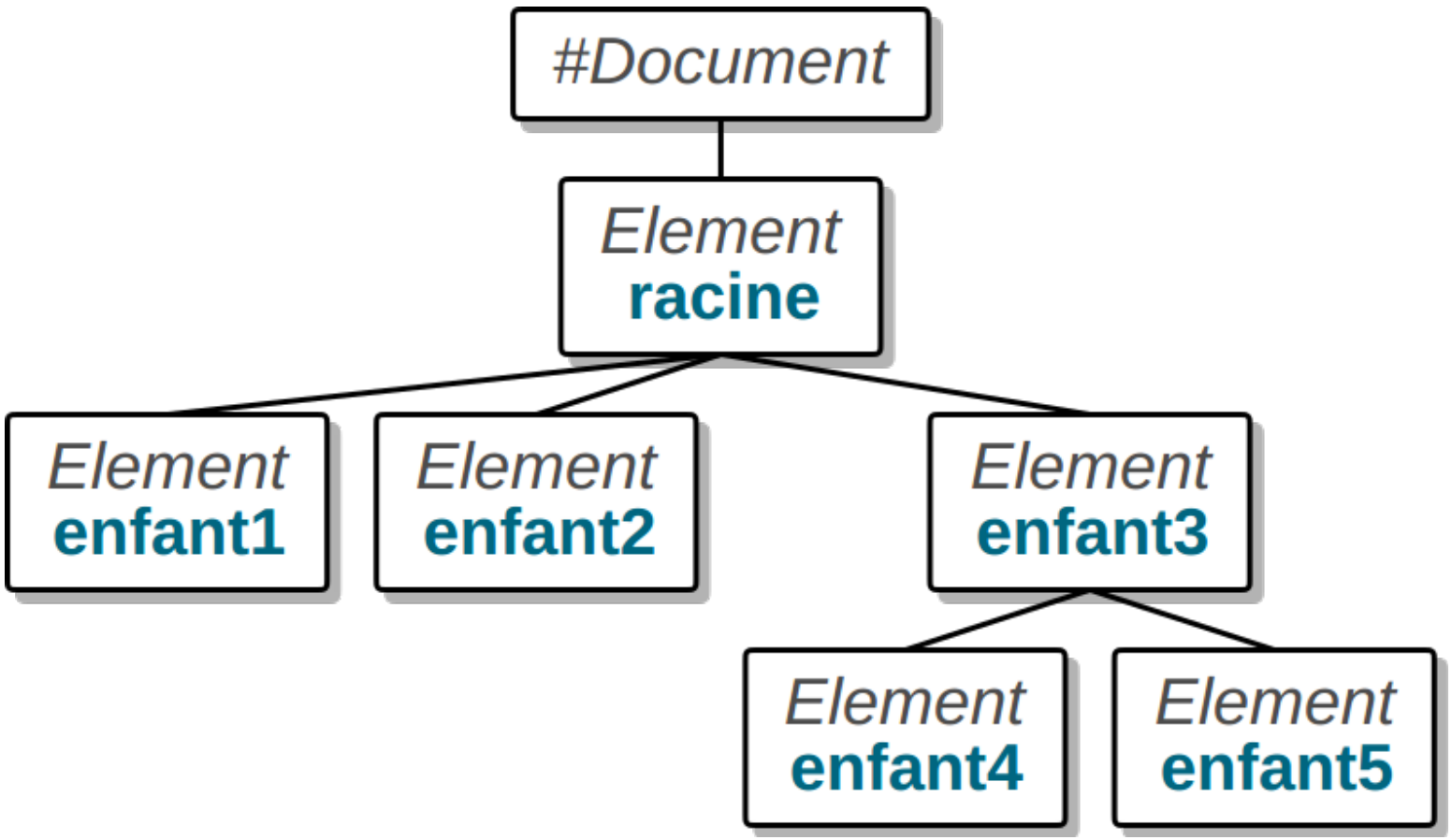
nœuds `#text` ils représentent le texte situé entre deux balises. Les nœuds texte sont des feuilles dans l'arbre.

Notez que différents textes peuvent être entrelacés avec des éléments. Voir le cas de `<lieu>` dans le contenu de `<profession>`. Il est possible de le faire, mais ce n'est pas forcément souhaitable.

Les autres types de nœuds seront présentés au fur et à mesure.

Vocabulaire

Soit cet arbre XML :



Pierre Nerzic

Vocabulaire (suite)

Voici comment on désigne les différents nœuds les uns par rapport aux autres :

- `<racine>` est le nœud **parent** du nœud **enfant** (*child*) `<enfant3>`, lui-même parent de `<enfant4>` et `<enfant5>`,
- `<racine>`, `<enfant3>` sont des nœuds **ancêtres** (*ancestors*) de `<enfant4>` et `<enfant5>`,
- `<enfant4>` et `<enfant5>` sont des **descendants** (*descendants*) de `<racine>` et `<enfant3>`,
- `<enfant1>` est un nœud **frère** (*sibling*[1]) de `<enfant2>` et réciproquement.

[1] *siblings* = fratrie.

Détails du format XML

Pierre Nerzic

Prologue XML

La première ligne d'un document XML est appelée *prologue XML*. Elle spécifie la version de la norme XML utilisée (1.0 ou 1.1 qui sont très similaires[2]) ainsi que l'encodage :

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<?xml version="1.0" encoding="iso-8859-15"?>
```

[2] La norme 1.1 est plus claire concernant certains nouveaux caractères unicode qui peuvent spécifier des retours à la ligne et autres caractères de contrôle.

Norme Unicode

La norme Unicode définit un ensemble abstrait de plus de 245000 caractères représentable sur un ordinateur, par exemple é, €, «, ©. . . Ces caractères doivent être codés sous forme d'octets. C'est là qu'il y a plusieurs normes :

- ASCII ne représente que les 128 premiers caractères Unicode.
- ISO 8859-1 ou Latin-1 représente 191 caractères de l'alphabet latin n°1 (ouest de l'europe) à l'aide d'un seul octet. Les caractères € et œ n'en font pas partie, pourtant il y a æ.
- ISO 8859-15 est une norme mieux adaptée au français. Elle rajoute € et œ et supprime des caractères peu utiles comme ¤.
- UTF-8 représente les caractères à l'aide d'un nombre variable d'octets, de 1 à 4 selon le caractère. Par exemple : A est codé par `0x41`, é par `0xC3,0xA9` et € par `0xE2,0x82,0xAC`.

Commentaires XML

On peut placer des commentaires à peu près partout dans un document XML. La syntaxe est identique à celle d'un fichier HTML. Un commentaire peut d'étendre sur plusieurs lignes. La seule contrainte est de ne pas pouvoir employer les caractères -- dans le commentaire, même s'ils ne sont pas suivis de >

```
<!-- voici un commentaire -->
ceci n'est pas un commentaire
<!--
    voici un autre commentaire
    et ça -- , c'est une erreur
-->
```


Pierre Nerzic

Attention aux -- dans les commentaires

En fait, en XML (comme en SGML), `<!--` et `-->` ne sont pas des marques de commentaires comme `/*` et `*/` en C.

- `<!` marque le début d'une instruction de traitement destinée à l'analyseur
- `>` signale la fin de cette instruction de traitement
- `--` inverse le mode « commentaire », entre « hors commentaire » et « dans un commentaire ».

Ainsi, un commentaire `<!--commentaire-->` est analysé ainsi :

1. `<!` début d'une instruction de traitement
2. `--` passage en mode commentaire
3. `commentaire` : ignoré, on peut donc tout y mettre sauf `--`
4. `--` sortie du mode commentaire, ne rien mettre après
5. `>` sortie de l'instruction de traitement

Options après le prologue

Après le prologue, on peut trouver plusieurs parties optionnelles délimitées par <?...?> ou <!...>. Ce sont des instructions de traitement, des directives pour l'analyseur XML.

Par exemple :

- un *Document Type Definitions* (DTD) qui permet de valider le contenu du document. C'est l'objet du prochain cours.
- une feuille de style, voir cours 4.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personne SYSTEM "personne.dtd">
<?xml-stylesheet href="personne.css" type="text/css"?>
<personne>
    ...
</personne>
```


Pierre Nerzic

Éléments

Les éléments définissent la structure du document. Un élément est délimité par :

- une balise ouvrante `<nom attributs...>`
- une balise fermante `</nom>` obligatoire.

Le contenu de l'élément se trouve entre les deux balises. Ce sont des éléments enfants et/ou du texte.

```
<parent>
    texte1
    <enfant>texte2</enfant>
    texte3
</parent>
```

Dans le cas où l'élément n'a pas de contenu (*élément vide*), on peut regrouper ses deux balises en une seule `<nom attributs.../>`

Choses interdites

Les règles d'imbrication XML interdisent différentes configurations qui sont plus ou moins tolérées en HTML :

- plusieurs racines dans le document,
- des éléments non terminés (NB: XML est sensible à la casse),
- des éléments qui se chevauchent.

```
<element1>
    <element2>
    </Element2>
    <element3>
    </element1>
</element3>
```

En XML, cela crée des erreurs « *document mal formé* ».

Pierre Nerzic

Choses permises

Parmi les choses permises, le même type d'élément peut se trouver plusieurs fois avec le même parent, avec des contenus identiques ou différents (à vous de définir si ça a du sens) :

```
<element1>
    <element2>texte1</element2>
    <element2>texte2</element2>
    <element2>texte1</element2>
</element1>
```

Il est possible d'interdire cela, cependant ce n'est pas relatif à la syntaxe XML mais à la validation du document par rapport à une spécification, et aussi ce qu'on veut faire avec le document. Voir les DTD et les schémas dans le prochain cours.

Pierre Nerzic

Noms des éléments

Les noms des éléments peuvent employer de nombreux caractères Unicode (correspondant au codage déclaré dans le prologue) mais pas les signes de ponctuation.

- L'initiale doit être une lettre ou le signe `_`
- ensuite, on peut trouver des lettres, des chiffres ou `-` `.` `_`

Le caractère `:` permet de séparer le nom en deux parties : préfixe et *nom local*. Le tout s'appelle *nom qualifié*. Par exemple `iut:departement` est un nom qualifié préfixé par `iut`. Ce préfixe permet de définir un *espace de nommage*.

$$\textit{nom qualifié} = \textit{préfixe} : \textit{nom local}$$

Espaces de nommage

Un espace de nommage (*namespace*) définit une famille de noms afin d'éviter les confusions entre des éléments qui auraient le même nom mais pas le même sens. Cela arrive quand le document XML modélise les informations de plusieurs domaines.

Voici un exemple dans le domaine de la vente de meubles. Le document modélise une table (avec 4 pieds) et aussi un tableau HTML pour afficher ses dimensions. On voit la confusion.

```
<meuble id="765">
  <table prix="74,99€">acajou</table>
  <table border="1">
    <tr><th>longueur</th><th>largeur</th></tr>
    <tr><td>120cm</td><td>80cm</td></tr>
  </table>
</meuble>
```


Pierre Nerzic

Remarques

Le document de l'exemple précédent modélise une situation particulière. On veut que le document contienne à la fois une description en XML et du code HTML. Sans doute, la partie HTML sera extraite et affichée à un utilisateur.

C'est finalement ce qu'on fait du document, son utilisation, ses transformations qui guident sa construction. Nous verrons tout ce qui concerne la transformation d'un document XML en semaine 4.

L'exemple précédent est un peu artificiel puisqu'on a fait exprès de choisir le même nom pour des éléments de nature différente. On verra dans un prochain cours qu'il est pratique d'employer des espaces de nommage quand on mélange au sein du même document XML des instructions de traitement et des données. C'est le cas dans les feuilles de style XSLT.

Pierre Nerzic

Évolution des normes

Une autre raison pour employer des espaces de nommage est de permettre l'extension d'une norme qui risque d'évoluer par la suite. Par exemple la norme GPX qui définit des fichiers de géolocalisation des GPS. Elle définit par exemple la structure d'un point de trace `<trkpt>`. Il a un certain nombre d'attributs comme `lon` et `lat` pour les coordonnées géographiques, et de sous-élements tels que `<ele>` qui mémorisent l'altitude d'un point.

La société Garmin a rajouté de nouvelles informations pour les sportifs tels que `<hr>` pour mémoriser le rythme cardiaque. Or il y a le risque que dans l'avenir, le format GPX évolue et définisse de son côté un élément `<hr>` avec une autre signification.

Grâce à un espace de nommage spécifique, la société Garmin peut définir ses propres balises `<garmin:hr>` sans risquer de conflit avec la norme GPX générale.

Définition d'un espace de nommage

On doit choisir un URI, par exemple un URL, pour identifier l'espace de nommage. Cet URI n'a pas besoin de correspondre à un véritable contenu car il ne sera pas téléchargé.

Un URI est la généralisation d'un URL :

`schéma:[//[user:passwd@]hôte[:port]][/]chemin[?requête][#fragment]`,

par exemple

`http://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Syntax`.

Les URN sont au format `urn:NID:NSS`, ex: `urn:iutlan:xmlsem1`

Ensuite on rajoute un attribut spécial à la racine du document :

```
<?xml version="1.0" encoding="utf-8"?>
<racine xmlns:PREFIXE="URI">
    <PREFIXE:balise>...</PREFIXE:balise>
</racine>
```

Exemple revu

Voici l'exemple précédent, avec deux *namespaces*, un URN pour les meubles et un URL pour HTML :

```
<?xml version="1.0" encoding="utf-8"?>
<meuble:meuble id="765"
    xmlns:meuble="urn:iutlan:meubles"
    xmlns:html="http://www.w3.org">
  <meuble:table prix="74,99€">acajou</meuble:table>
  <html:table border="1">
    <html:tr><html:th>longueur</html:th>...</html:tr>
    <html:tr><html:td>120cm</html:td>...</html:tr>
  </html:table>
</meuble:meuble>
```

Notez le préfixe également appliqué à la racine du document.

Pierre Nerzic

Namespace par défaut

Lorsque la racine du document définit un attribut `xmlns="URI"`, alors par défaut toutes les balises du document sont placées dans ce namespace, donc pas besoin de mettre un préfixe.

```
<?xml version="1.0" encoding="utf-8"?>
<book xmlns="http://docbook.org/ns/docbook">
```

Ça peut s'appliquer également localement à un élément et ça concerne toute sa descendance :

```
<meuble id="765" xmlns="urn:iutlan:meubles">
  <table prix="74,99€">acajou</table>
  <table border="1" xmlns="http://www.w3.org">
    <tr><th>longueur</th>...</tr>
    <tr><td>120cm</td>...</tr>
  </table>
</meuble>
```

Namespace par défaut, attributs et valeurs

Définir un *namespace* par défaut associe seulement les éléments à ce *namespace*, mais pas leurs attributs.

Les attributs qui n'ont pas de préfixe n'ont pas de *namespace*. L'interprétation des attributs dépend de l'élément dans lequel ils se trouvent. Voir cette réponse sur StackOverflow.

Cependant, pour clarifier la situation, on rajoute de préférence un préfixe devant les attributs. Par exemple, `android:layout_width`. Avec le préfixe, il n'y a plus d'ambiguïté possible : l'attribut appartient au *namespace* concerné.

On retrouve le même problème avec les valeurs de certains attributs, voir les schémas XML dans le cours 3.

Pierre Nerzic

Attributs

Les attributs caractérisent un élément. Ce sont des couples `nom="valeur"` ou `nom='valeur'`. Ils sont placés dans la balise ouvrante.

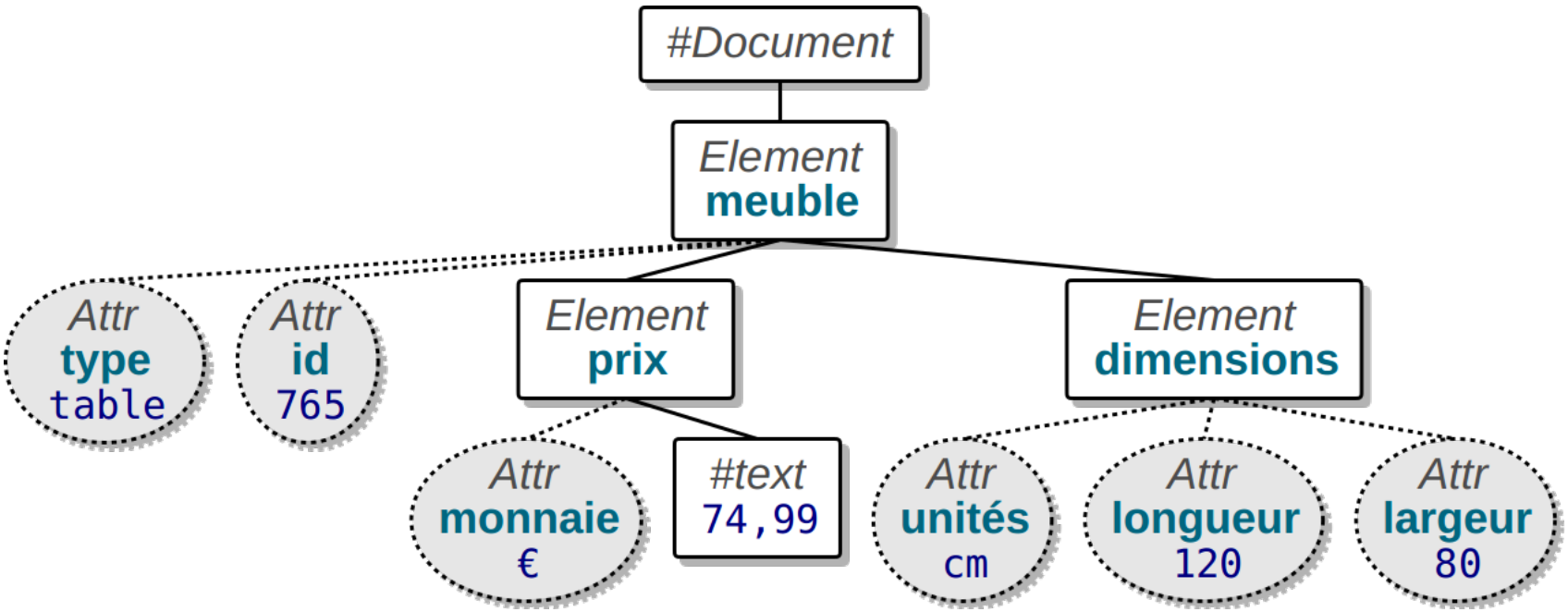
```
<?xml version="1.0" encoding="utf-8"?>
<meuble id="765" type='table'>
    <prix monnaie='€'>74,99</prix>
    <dimensions unites="cm" longueur="120" largeur="80"/>
    <description langue='fr'>Belle petite table</description>
</meuble>
```

Remarques :

- Il n'y a pas d'ordre entre les attributs d'un élément,
- Un attribut ne peut être présent qu'une fois.

Attributs dans l'arbre du document

Les attributs sont vus comme des nœuds dans l'arbre sous-jacent.
Voici l'arbre d'une partie de l'exemple précédent :



Pierre Nerzic

Entités

Certains caractères sont interdits dans le contenu des éléments. Comme il est interdit de les employer, XML propose une représentation appelée *référence d'entité* ou *entité* pour simplifier. On retrouve le même concept en HTML.

Caractère	Entité
<	`<`
>	`>`
&	`&`
'	`&apos;`
"	`&quot;`

Exemple

Voici un exemple d'emploi d'entités :

```
<?xml version="1.0" encoding="utf-8"?>
<achat-si type-groupement="&amp;&amp;">
  <test>prix &lt; 50.00</test>
  <test>description &lt;&gt; <str value="&quot;&quot;"/></test>
</achat-si>
```

- Cela entraîne une erreur si on remplace ces entités par le caractère correspondant,
- Ces entités sont automatiquement remplacées par les caractères lorsqu'on traite le fichier.

NB: ce contenu bizarre vise uniquement à illustrer l'emploi d'entités.

Pierre Nerzic

Entités définies dans le document

Il est possible de créer ses propres entités. Dans ce cas, elles peuvent représenter beaucoup plus qu'un caractère, toute une chaîne, voire même tout un arbre XML.

Voici comment faire. Il faut définir un nœud `DOCTYPE` appelé *Document Type Definitions* (DTD) concernant la racine du document. Normalement, on y rajoute les règles indiquant la structure du document XML, mais ici, il n'y a que les entités.

```
<!DOCTYPE racine [
    <!ENTITY nom "texte à mettre à la place">
]>
```

L'entité `&nom;` est appelée *entité interne*.

Il faut aussi rajouter `standalone="no"` dans le prologue du document.

Exemple d'entité interne

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE personne [
    <!ENTITY adresse "IUT de Lannion - Dept Informatique">
    <!ENTITY cours   "Cours XML">
]>
<personne>
    <lieu>&adresse;</lieu>
    <role>&cours;</role>
</personne>
```

Pour afficher le résultat, on peut l'ouvrir dans Firefox ou utiliser `xmllint` :

```
xmllint --noent exemple.xml
```

Entités externes

Il s'agit d'une entité qui représente tout un document XML contenu dans un autre fichier. Voici comment on la définit :

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE racine [
    <!ENTITY voiture1 SYSTEM "voiture1.xml">
]>
<garage>
    <vente>&voiture1;</vente>
</garage>
```

- Le mot clé `SYSTEM` commande la lecture d'un URL, ici, c'est un fichier local
- Le contenu du document `voiture1.xml` est inséré à la place de l'entité `&voiture1;`

Texte

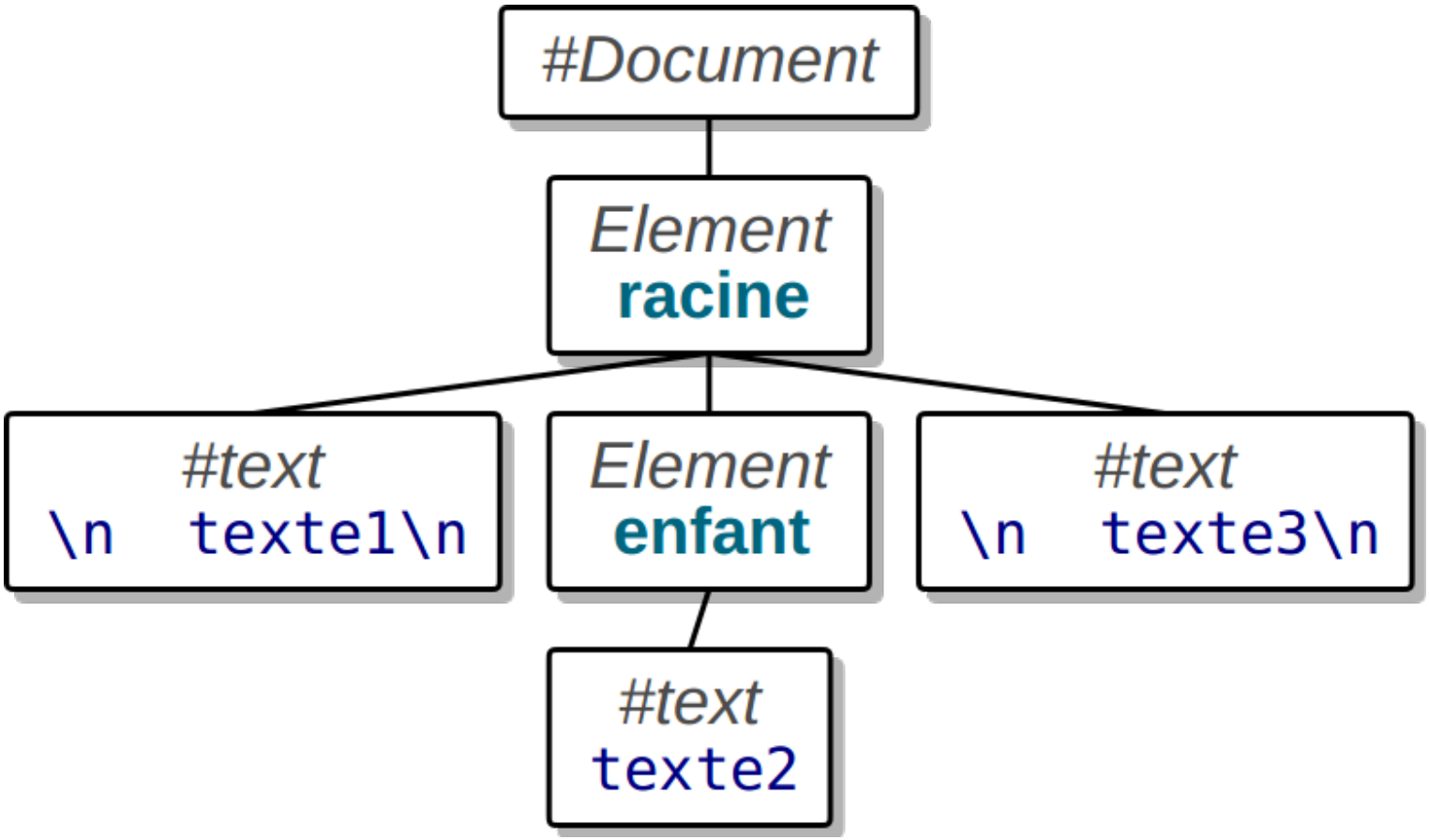
Les textes font partie du contenu des éléments et sont vus comme des nœuds enfants. Il faut bien comprendre que la totalité du texte situé entre les balises, y compris les espaces et retour à la ligne font partie du texte.

```
<?xml version="1.0" encoding="utf-8"?>
<racine>
  texte1
  <enfant>texte2</enfant>
  texte3
</racine>
```


Pierre Nerzic

Arbre correspondant

Voici l'arbre correspondant à l'exemple précédent. Notez les retours à la ligne et espaces présents dans les textes sauf `texte2`.



Pierre Nerzic

Sections CDATA

Lorsqu'on veut écrire du texte brut à ne pas analyser en tant qu'XML, on emploie une section CDATA : 

```
<?xml version="1.0" encoding="utf-8"?>
<fichier>
    <nom>exemple1.xml</nom>
    <md5><![CDATA[19573b741c0c5c8190a83430967bfa58]]></md5>
</fichier>
```

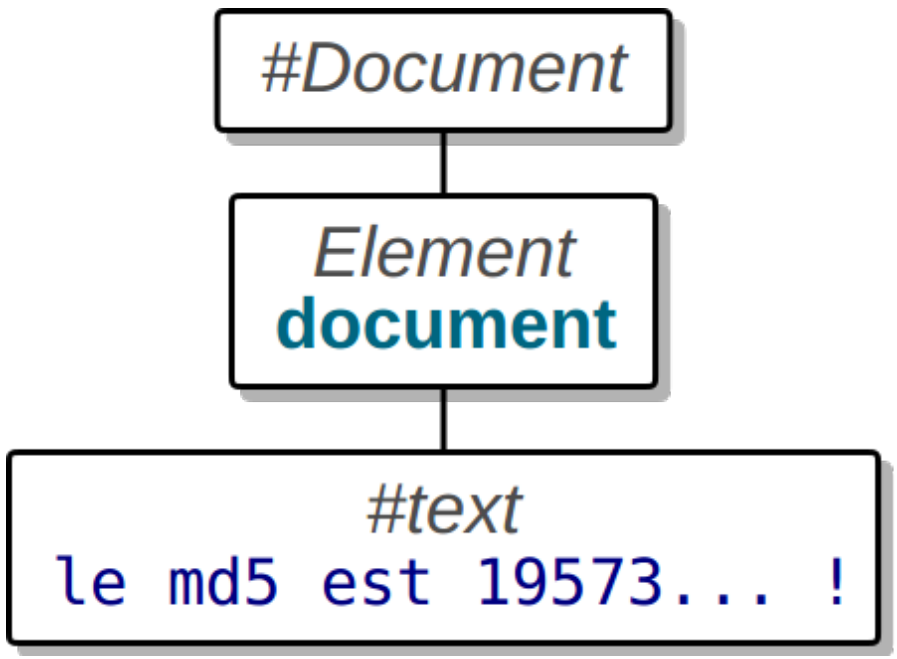
La partie entre `<![CDATA[` et `]]>` est ignorée par les analyseurs XML, on peut mettre n'importe quoi sauf `]]>`. Ces données sont considérées comme du texte par les analyseur.

Utiliser `xmllint --nocdata document.xml` pour le voir sans les marqueurs.

Fusion des CDATA et textes

S'il y a des textes avant ou après une section CDATA, ils sont tous fusionnés avec elle.

```
<?xml version="1.0" encoding="utf-8"?>
<document>le md5 est <![CDATA[19573...]]> !</document>
```



Pierre Nerzic

Modélisation

Pierre Nerzic

Modélisation d'un TE

XML sert à représenter des informations. Regardons comment cela se passe pour un modèle entité-association, en commençant par la modélisation d'un TE, puis d'un TA en XML.

Pour un TE, il est simple de les représenter à l'aide d'éléments.

Voici par exemple une liste de voitures, chacune ayant un identifiant, une marque et une couleur :

```
<?xml version="1.0" encoding="utf-8"?>
<voitures>
  <voiture id="125" marque="Renault" couleur="grise"/>
  <voiture id="982" marque="Peugeot" couleur="noire"/>
</voitures>
```


Pierre Nerzic

Propriétés dans les attributs ou des sous-éléments ?

En ce qui concerne les propriétés de l'entité, on peut les placer :

- dans les attributs des éléments :

```
<voiture id="125" marque="Renault" couleur="grise"/>
```

- en tant que sous-éléments :

```
<voiture>
  <id>125</id>
  <marque>Renault</marque>
  <couleur>grise</couleur>
</voiture>
```

- ou employer un mélange des deux.

Ces choix dépendent principalement des traitements qu'on souhaite effectuer sur le document ; voir le cours de la semaine 4.

Pierre Nerzic

Attributs ou sous-éléments ? (suite)

Dans la réflexion pour choisir de représenter une information sous forme d'attributs ou de sous-éléments, il faut prendre en compte ces critères :

- Les attributs ne peuvent pas contenir plusieurs valeurs, tandis qu'on peut avoir plusieurs sous-éléments ayant le même nom et des contenus différents,
- Les attributs ne peuvent pas contenir de structures d'arbres, au contraire des éléments,
- Les attributs ne sont pas facilement extensibles, tandis qu'on peut toujours rajouter de nouveaux sous-éléments dans une hiérarchie sans changer les logiciels existants,
- Il n'est pas plus compliqué d'accéder à des sous-éléments qu'à des attributs car tous sont des nœuds dans l'arbre sous-jacent.

Attributs ou contenu ? (fin)

Avec les remarques précédentes, il est possible de construire ce genre de document :

```
<voiture id="649">
  <marque niveau="filiale">Dacia</marque>
  <marque niveau="generale">Renault</marque>
  <couleur>grise
    <nuance reference="EYP">gris banquise</nuance>
  </couleur>
</voiture>
```

Remarque : cela dépasse ce qu'on peut représenter dans une table de base de données. Ce serait le résultat d'une jointure

Pierre Nerzic

Associations

La représentation XML d'une association entre deux TE est un peu problématique quand les cardinalités ne sont pas (1,1) d'un côté. Les documents XML sont essentiellement hiérarchiques et représentent les informations uniquement vues d'une seule chose. Il faudrait donc mettre l'extension de la relation.

Lorsque l'une des cardinalités est (1,1), il est possible de construire une hiérarchie centrée sur l'un des TE. Voici par exemple, une relation de location entre (1,1) véhicule et *0* à *n* clients :

```
<voiture id="649">
  <marque>Renault</marque> ...
  <location date="15/11/2021"><nom>Dupond</nom> ...</location>
  <location date="25/01/2022"><nom>Dupont</nom> ...</location>
</voiture>
```