

Données ouvertes : traitement, formats

Guillaume Allègre
Guillaume.Allegre@silecs.info

Polytech Grenoble - RICH

2017-2018

Traitements des données

Workflow de données

Récupération des données brutes

- ▶ fichier (éventuellement prétraitement manuel)
- ▶ API web : récupération au vol
- ▶ autant que de sources de données

"Traitement"

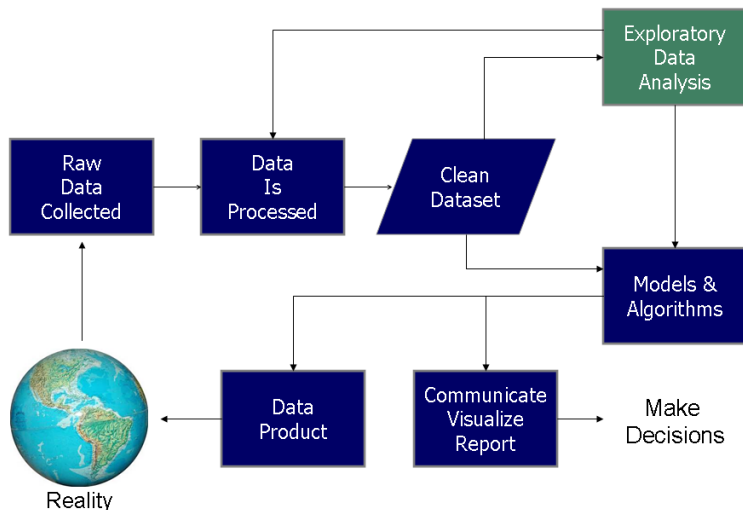
- ▶ Calculs... (cf suite)

Publication des données raffinées

- ▶ format final simple (ex. table HTML)
- ▶ mise à disposition d'un webservice (ex. index/recherche)
- ▶ visualisation ("*datavisualization*" ou "*dataviz*")

Workflow de données - 2

Data Science Process



Exemples de traitements - 1

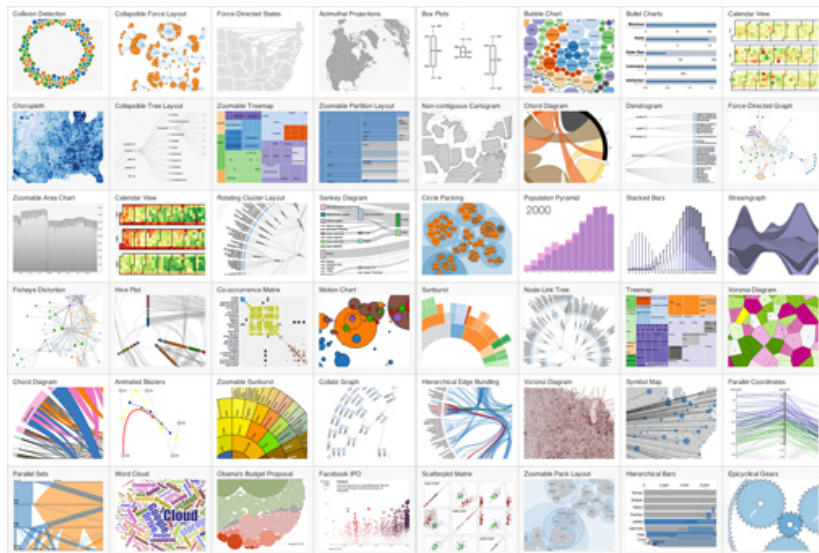
Agrégation

- ▶ ex. liste des communes de France, dont population, superficie
- ▶ statistiques : moyenne, médiane, quartiles...
- ▶ top N : les plus peuplées, les plus grandes...
- ▶ domaine des **statistiques descriptives**

Visualisation

- ▶ stats+ : courbes, histogrammes, diagrammes de classes
- ▶ dynamique : côté serveur / clients
- ▶ cartographie (choroplète, monde, pays, ville...)

Visualisations



Exemples de traitements - 2 Mashups

Mashups

- ▶ utilisations de plusieurs sources de données
- ▶ de nature identiques ou différentes
- ▶ exemple
 - ▶ localisation des crèches et écoles (ex. data.metropolegrenoble)
 - ▶ tracés des routes par importance (ex. OpenStreetMap)
 - ▶ relevés qualité de l'air (ex. Air Rhône-Alpes)

Précautions

- ▶ Cohérence des données (ex. date des collectes)
- ▶ "*jointure*" des données : utilisation d'un identifiant commun
- ▶ ex. code INSEE pour les communes (ex. 38421 Saint-Martin-d'Hères)

Traitements des données

Échanges

- ▶ fichiers : **formats** et systèmes de fichiers
- ▶ réseau : **protocoles**

Exploitation des données

- ▶ bases de données
 - ▶ relationnelles (SQL) : Oracle, MySQL/MariaDB, PostgreSQL
 - ▶ bases "NoSQL" : orientées document ou clé-valeur
- ▶ langage et structures de données
- ▶ (ici PHP)

Rappels : les tableaux en PHP

Pourquoi utiliser des tableaux ?

```
$val1 = "Moi";  
$val2 = "Toi";  
$val3 = "Lui";  
echo $val1;
```

Quand on a plusieurs valeurs reliées, mieux vaut les mettre dans un tableau.

```
$val = array("Moi", "Toi", "Lui");  
echo $val[0];
```

Les tableaux numériques

- ▶ `array(...)` permet de créer un tableau PHP.
- ▶ ou `[...]` à partir de PHP 5.4.
- ▶ Par défaut, les tableaux sont numérotés à partir de 0.
- ▶ Pour accéder à un élément : `$val[...]`
- ▶ Pour ajouter à la fin du tableau :

```
$val[] = array("Nous"); # $val[3]
```

Rappel : les tableaux en PHP 2

- ▶ plus précisément : **tableaux associatifs ordonnés**
- ▶ paires **clé / valeur** (dictionnaire) :

```
$tab = array(  
    "yes" => "oui",  
    "no"  => "non",  
);
```

- ▶ **accès aux valeurs** :

```
// Lecture  
echo "Traduction de yes : " . $tab["yes"];  
// Ecriture  
$tab["never"] = "jamais";
```

- ▶ multitude de fonctions pour les tableaux :

<http://fr.php.net/manual/fr/function.array.php>

Rappel : parcours d'un tableau

Pour ne lire que le contenu des cases

```
<?php
foreach ($tableau as $valeur) {
    echo "$valeur\n";
}
```

Pour lire à la fois la clé de la case et sa valeur

```
<?php
foreach ($tableau as $cle => $valeur) {
    echo "$cle -> $valeur\n";
}
```

foreach est la boucle la plus fréquente en PHP !

Ex. d'utilisation d'un tableau structuré

```
$annuaire = array( [  
    'nom' => 'Allegre',  
    'prenom' => 'Guillaume',  
    'email' => 'allegre.guillaume@free.fr',  
    'role' => 'enseignant'],  
[  
    'nom' => 'Pittion',  
    'prenom' => 'Sebastien',  
    'email' => 'sebastien.pittion@exemple.fr',  
    'role' => 'enseignant'],  
[  
    'nom' => 'Dupont',  
    'prenom' => 'Jeanne',  
    'email' => 'jeanne.dupont@e.univ-grenoble-alpes.fr',  
    'role' => 'etudiant'],  
);  
print_r($annuaire);
```

Ex. d'utilisation d'un tableau structuré - résultat `print_r`

```
Array
(
    [0] => Array
        (
            [nom] => Allegre
            [prenom] => Guillaume
            [email] => allegre.guillaume@free.fr
            [role] => enseignant
        )
    [1] => Array
        (
            [nom] => Pittion
            [prenom] => Sebastien
            [email] => sebastien.pittion@exemple.fr
            [role] => enseignant
        )
    [2] => Array
        (
            [nom] => Dupont
```

Variante plus structurée

```
$annuaire2 = array( [  
    'etatscivil' => ['nom' => 'Allegre',  
                     'prenom' => 'Guillaume',  
                     'dnaiss' => '19740928'],  
    'email' => 'allegre.guillaume@free.fr',  
    'role' => 'enseignant' ],  
[  
    'etatscivil' => ['nom' => 'Pittion',  
                     'prenom' => 'Sebastien',  
                     'dnaiss' => '00000000'],  
    'email' => 'sebastien.pittion@exemple.fr',  
    'role' => 'enseignant' ],  
[  
    'etatscivil' => ['nom' => 'Dupont',  
                     'prenom' => 'Jeanne',  
                     'dnaiss' => '19940412'],  
    'email' => 'jeanne.dupont@e.univ-grenoble-alpes.fr',  
    'role' => 'etudiant' ]  
);
```

Variante plus structurée - résultat `print_r`

```
Array
(
    [0] => Array
        (
            [etatcivil] => Array
                (
                    [nom] => Allegre
                    [prenom] => Guillaume
                    [dnaiss] => 19740928
                )
            [email] => allegre.guillaume@free.fr
            [role] => enseignant
        )
    [1] => Array
        (
            [etatcivil] => Array
                (
                    [nom] => Pittion
                    [prenom] => Sebastien
                )
        )
)
```

Utilisation d'une classe - validation des données

```
class EntreeAnnuaire extends stdClass
{
    public $nom;
    public $prenom;
    public $email;
    public $role;

    public function validate()
    {
        ...
    }
}
```


Principaux formats de données

Identifier les formats : MIME

Comment déterminer un type de fichiers ?

- ▶ Plusieurs concepts à distinguer
 - ▶ l'extension du fichier (si elle existe) : métadonnée (ex. `data.csv`)
 - ▶ sa signature (si elle existe)
 - ▶ son type MIME (Multipurpose Internet Mail Extensions)
 - ▶ `Content-Type: text/plain; charset=UTF-8`
 - ▶ protocoles : mail (SMTP), `HTTP`, HTTPS...
 - ▶ les applications le prenant en charge
- ▶ Techniquement
 - ▶ ex. `Content-Type: application/json`
 - ▶ ex. `Content-Type: text/csv, text/html`
 - ▶ `file -i` renvoie le type MIME
 - ▶ `/etc/mime.types`, `/etc/mime-magic`

Principaux formats de données

CSV - comma separated values

- ▶ fichier tabulé, texte

JSON - JavaScript Object Notation

- ▶ format récent, orienté web
- ▶ proche des structures des données
- ▶ formats dérivés : GeoJson, TopoJson...

XML - eXtended Markup Language

- ▶ format très extensible et versatile
- ▶ très nombreux formats dérivés
- ▶ bien plus lourd que JSON

Formats métiers spécifiques

- ▶ ...

CSV et autres formats tableur

CSV - Comma separated values

Le format CSV

- ▶ format **tabulé** simple
- ▶ lisible à l'oeil humain
- ▶ *parsable* assez facilement
- ▶ **variantes** : séparateur (tab, virgule, point-virgule), guillemets

En pratique

- ▶ prototypage rapide en shell (ligne de commande Unix)
- ▶ lisible dans un tableur (LibreOffice...)
- ▶ en PHP, lecture : `fgetcsv()`, `str_getcsv()`
- ▶ écriture `fputcsv()`

CSV en PHP

Exemple de données

`borneswifi_EPSG4326.csv`

```
AP_ANTENNE1,Antenne 1,longitude,latitude
AP_APP_GAGNANT,Musée Stendhal Appt Gagnon,5.7280,45.1915
AP_BIB_ABBAYE,Bibliothèque Abbaye,5.7420,45.1794
AP_BIB_ALLIANCE,Bibliothèque Alliance,5.7245,45.1733
AP_BIB_ARLEQUIN,Bibliothèque Arlequin,5.7329,45.1640
AP_BIB_CENTRE_VILLE,Bibliothèque Centre Ville,5.7293,45.1903
AP_BIB_CENTRE_VILLE2,Bibliothèque Centre Ville,5.7293,45.1903
[...]
```

Code : 3 fonctions PHP dédiées

- ▶ `fputcsv` formate une ligne en CSV et l'écrit dans un fichier
- ▶ `str_getcsv` analyse une chaîne de caractères CSV dans un tableau
- ▶ `fgetcsv` obtient une ligne depuis un pointeur de fichier et l'analyse pour des champs CSV

fgetcsv par l'exemple

```
$row = 1;
if (($handle = fopen("test.csv", "r")) !== FALSE) {
    while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
        $num = count($data);
        echo "<p> $num champs sur la ligne $row: <br /></p>\n";
        $row++;
        for ($c=0; $c < $num; $c++) {
            echo $data[$c] . "<br />\n";
        }
    }
    fclose($handle);
}
```

Prototypage CSV - filtres Unix

`ls -l | wc` sortie de `ls` canalisée vers l'entrée du filtre `wc`.

`find /etc |& wc` StdOut et StdErr fusionnées puis canalisées

Exemples

1. `cat` taper `Ctrl+D` = fin de flux
2. `cat liste.txt | wc -l`
3. `wc -l liste.txt`
4. `wc -l < liste.txt`
5. `cat < liste.txt | wc -l`
6. `wc -l liste.txt 12.txt 13.txt`
7. `cat liste.txt 12.txt 13.txt | wc -l`

Filtres textes courants

Principe Unix : une tâche, un outil.

Beaucoup de filtres fonctionnent ligne par ligne :

- ▶ **grep** Garde les lignes correspondant à un motif (cf. **WHERE**)
Ex. `ls / | grep v`
- ▶ **cut** Conserve les colonnes données (cf. **SELECT**)
- ▶ **head** Premières lignes
- ▶ **tail** Dernières lignes
- ▶ **sort** Trie les lignes
- ▶ **uniq** Enlève les doublons
- ▶ moins courants : **tr**, **tac**, **paste**, **fmt**...
- ▶ paquet **coreutils**

Exemple complet

En TP !

Les formats tableur

- ▶ **XLS** Excel traditionnel
- ▶ **XLSX** Excel en XML (récent)
- ▶ **ODS** OpenDocument Spreadsheet (LibreOffice...) (XML)

Avantages

- ▶ permettent les métadonnées
- ▶ permettent la mise en forme (lecture humaine)

Inconvénients

- ▶ incompatibles avec les filtres unix
- ▶ nécessitent des bibliothèques spécifiques (courantes)
- ▶ plusieurs versions / peuvent évoluer

Les formats JSON

JSON

Le format JSON

- ▶ format de **sérialisation** de données structurées (tableaux, objets...)
- ▶ produit une chaîne de caractères : lisible
- ▶ *parsable* par une machine
- ▶ provient de JavaScript : JavaScript Object Notation
- ▶ intégré à tous les langages modernes

Usages

- ▶ sérialisation/désérialisation de structures mémoire
- ▶ échanges de données
- ▶ fichiers de configuration (sans commentaires)

JSON - les types

Les scalaires

- ▶ chaîne de caractères, entourée de guillemets
- ▶ nombre : un nombre décimal signé
- ▶ booléen : True ou False
- ▶ Null

Les types complexes

- ▶ Tableau (à index numérique)
- ▶ Object, incluant les tableaux associatifs

JSON en pratique

En pratique

- ▶ en PHP : `json_encode()` et `json_decode()`
- ▶ en JavaScript : `JSON.parse()`

Variantes

- ▶ HJSON "Human JSON" accepte les commentaires
- ▶ BSON "Binary JSON" utilisé par MongoDB
- ▶ JSONP "JSON with Padding" : JavaScript inter-sites (ajax...)

echo json_encode(\$annuaire)

```
["nom" : "Allegre", "prenom" : "Guillaume", "email" : "allegre.guillaume@free.grenoble-alpes.fr", "role" : "etudiant"]
```

```
[  
  {  
    "nom" : "Allegre",  
    "role" : "enseignant",  
    "prenom" : "Guillaume",  
    "email" : "allegre.guillaume@free.fr"  
  },  
  {  
    "nom" : "Pittion",  
    "role" : "enseignant",  
    "prenom" : "Sebastien",  
    "email" : "sebastien.pittion@free.fr"  
  },  
  {  
    "nom" : "Dupont",  
    "role" : "enseignant",  
    "prenom" : "Jean",  
    "email" : "jean.dupont@free.fr"  }  
]
```


GeoJSON : un exemple de spécialisation de JSON

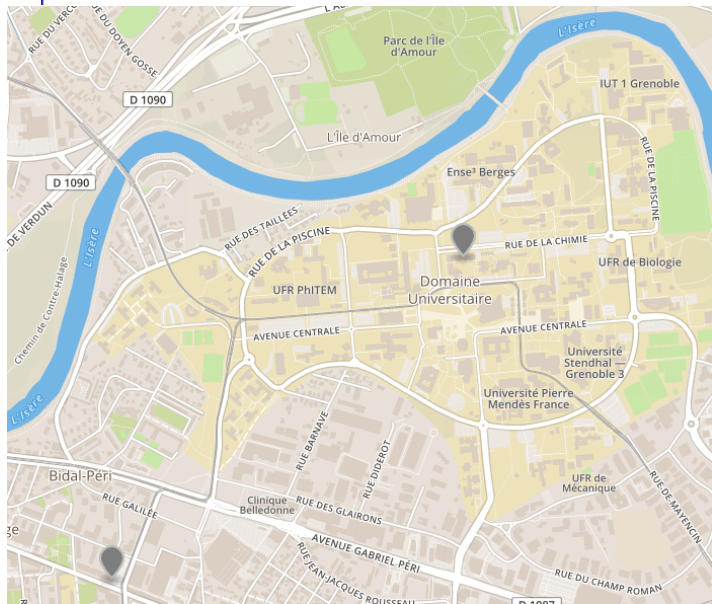
Format spécifique de JSON

- ▶ format spécialisé pour les objets géolocalisés
- ▶ partie géométrique : coordonnées + tracé (**Point**, **LineString**, **Polygon**, MultiPoint, MultiLineString, MultiPolygon)
- ▶ partie propriétés libres (nom, description, style d'affichage...)
- ▶ 2016 normalisation RFC 7946 par l'IETF
- ▶ documentation sur <http://geojson.org>

Outils

- ▶ **Leaflet** bibliothèque cartographique javascript très simple
- ▶ services en ligne : <http://geojson.io> et <http://umap.openstreetmap.fr>

Exemple - 1



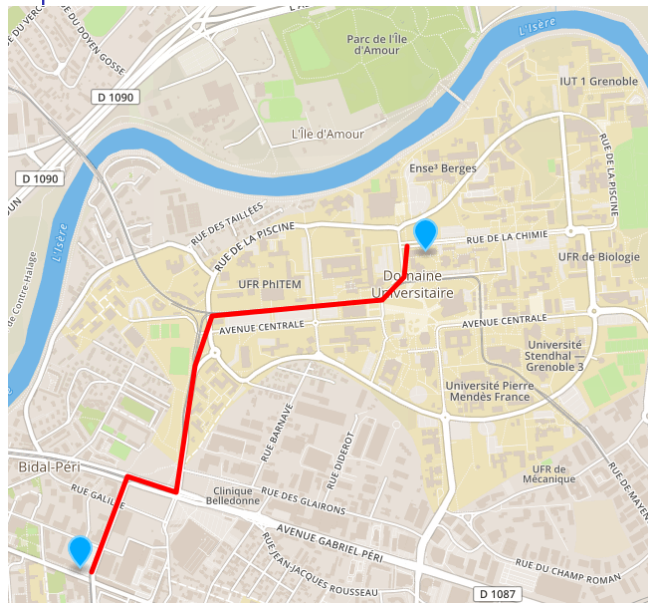
Exemple 1 deux points - 1/2

```
1  {
2    "type": " FeatureCollection ",
3    "features ": [
4      {
5        "type": "Feature",
6        " properties ": {
7          "marker-color": "#7e7e7e",
8          "name": "Polytech"
9        },
10       "geometry": {
11         "type": "Point",
12         "coordinates ": [ 5.75368, 45.18426 ]
13       }
14     },
```

Exemple 1 deux points - 2/2

```
15      {
16        "type": "Feature",
17        "properties": {
18          "marker-color": "#7e7e7e",
19          "name": "UFR IMA"
20        },
21        "geometry": {
22          "type": "Point",
23          "coordinates": [ 5.76857, 45.19387 ]
24        }
25      }
26    ]
27  }
```

Exemple - 2



Exemple 2 un itinéraire - 1/2

```
1  {
2    "type": " FeatureCollection ",
3    "features ": [
4      {
5        "type": "Feature",
6        " properties ": {
7          "marker-color": "#0af",
8          "name": "Polytech"
9        },
10       "geometry": {
11         "type": "Point",
12         " coordinates ": [ 5.75368, 45.18426 ]
13       }
14     },
15     {
16       "type": "Feature",
17       " properties ": {
18         "marker-color": "#0af",
19         "name": "UFR IMA"
```

Exemple 2 un itinéraire - 2/2

```
26      {
27        "type": "Feature",
28        "properties": {
29          "stroke": "#f00",
30          "stroke-width": "5",
31        },
32        "geometry": {
33          "type": "LineString",
34          "coordinates": [
35            [ 5.75417, 45.1841 ],
36            [ 5.75572, 45.1870 ],
37            [ 5.75782, 45.1865 ],
38            [ 5.75863, 45.1904 ],
39            [ 5.75936, 45.1919 ],
40            [ 5.76670, 45.1924 ],
41            [ 5.76765, 45.1931 ],
42            [ 5.76777, 45.1940 ]
43          ]
44        }

```

Les formats XML

XML - historique 1/2

GML - Generalized Markup Language

- ▶ IBM 1967-1969...
- ▶ Charles **G**odfarb, Edward **M**osher, Raymond **L**orie → GML
- ▶ langage de balisage pour documentation technique (juridique)
- ▶ renommé Generalized Markup Language

SGML - Standard Generalized Markup Language

- ▶ séparation complète et rigoureuse
 - ▶ document technique : texte balisé
 - ▶ structure DTD : Document Type Definition
 - ▶ feuille de style : présentation dépendant du support
- ▶ devient un standard de documentation technique industrielle
- ▶ impression, lecture écran, indexation/recherche
- ▶ 1986 normalisé ISO 8879 :1986

XML - historique 2/2

HTML - HyperText Markup Language

- ▶ application (sous-ensemble) de SGML, jusqu'à HTML5
- ▶ **Très** simplifiée
- ▶ 1990- créé par Tim Berners-Lee (avec le HTTP)
- ▶ nombreuses évolutions, standardisation par le W3C

XML - eXtensible Markup Language

- ▶ 1998, Jon Bosak (Sun Microsystems) et Dan Connolly (W3C)
- ▶ HTML → besoin d'un SGML simplifié + hypertexte/réseau
- ▶ sous-ensemble de SGML, simplifié
- ▶ mais fortement **extensible**
- ▶ nombreuses applications : XHTML, SVG, RSS, ODS, XLSX....

XML, un exemple

```
<?xml version="1.0" encoding="utf-8" ?>
<annuaire>
  <entree numero="1">
    <etatCivil>
      <nom>Allegre</nom>
      <prenom>Guillaume</prenom>
    </etatCivil>
    <email>allegre.guillaume@free.fr</email>
    <role>enseignant</role>
  </entree>
  <entree numero="2">
    <etatCivil>
      <nom>Pittion</nom>
      <prenom>Sebastien</prenom>
    </etatCivil>
    <email>sebastien.pittion@gmail.com</email>
    <role>enseignant</role>
  </entree>
</annuaire>
```

XML - les principes de base

Une généralisation du HTML

- ▶ structure d'arbre avec élément racine unique
- ▶ des éléments (balises ouvrantes/fermantes)
- ▶ des attributs
- ▶ des contenus libres (PCDATA)

Des différences

- ▶ élément racine **libre** (\neq `<html>`)
- ▶ des éléments libres
- ▶ des entités libres `&quelquechose;`
- ▶ ...

XML format "robuste" et générique

Deux niveaux de validation

- ▶ document **bien formé**
- ▶ document **conforme** à un schéma

Des outils standard

- ▶ nombreux éditeurs, dont `kate`, `xmlcopyeditor`
- ▶ en ligne : <http://utilities-online.info/xsdvalidation>
- ▶ `xmllint` en ligne de commande

Document bien formé?

Validation syntaxique

- ▶ déclaration XML obligatoire
- ▶ éléments sensibles à la casse
- ▶ valeurs d'attributs entre guillemets

Cohérence de l'arbre

- ▶ élément racine unique
- ▶ éléments fermés (balises ouvrates / fermantes)
- ▶ cohérence de la structure d'arbre

xmllint

xmllint

- ▶ paquet `libxml2-utils` (debian, ubuntu)
- ▶ documentation `man xmllint`
- ▶ `xmllint - -format` : réindente la sortie
- ▶ `xmllint - -noout` : supprime la sortie (uniquement le diagnostic)

XML en erreur ! (balise email)

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <annuaire>
3      <entree numero="1">
4          <etatCivil>
5              <nom>Allegre</nom>
6              <prenom>Guillaume</prenom>
7          </etatCivil>
8          <email>allegre.guillaume@free.fr<email>
9          <role>enseignant</role>
10     </entree>
11     <entree numero="2">
12         <etatCivil>
13             <nom>Pittion</nom>
14             <prenom>Sebastien</prenom>
15         </etatCivil>
16         <email>sebastien.pittion@gmail.com</email>
17         <role>enseignant</role>
18     </entree>
19 </annuaire>
```


xmllint - document mal formé

```
xmllint --noout annuaire.xml
```

```
annuaire1-err.xml:10: parser error :
```

```
  Opening and ending tag mismatch: email line 8 and entree
</entree>
```

```
^
```

```
annuaire1-err.xml:19: parser error :
```

```
  Opening and ending tag mismatch: email line 8 and annuaire
</annuaire>
```

```
^
```

```
annuaire1-err.xml:21: parser error :
```

```
  Premature end of data in tag entree line 3
```

```
^
```

```
annuaire1-err.xml:21: parser error :
```

```
  Premature end of data in tag annuaire line 2
```

```
^
```

Validation de DTD

Limitation de la validité syntaxique

- ▶ un document **bien formé** peut être **incohérent**
- ▶ ex. une entrée vide
- ▶ ex. `etapCivil` au lieu de `etatCivil`

DTD Document Type Definition

- ▶ une DTD est optionnelle
- ▶ elle permet de vérifier la **conformité** d'un document /fichier

Outils

- ▶ `[xmllint] --valid <fichier.xml>`
- ▶ ou `[xmllint] --dtdvalid <fichier.dtd> <fichier.xml>`

fichier annuaire.dtd

```
1  <!ELEMENT annuaire (entete?,entree+)>
2  <!ELEMENT entree (etatCivil, email, role, dateNaiss*)>
3    <!ATTLIST entree numero CDATA #REQUIRED>
4  <!ELEMENT etatCivil (nom, prenom)>
5  <!ELEMENT nom (#PCDATA)>
6  <!ELEMENT prenom (#PCDATA)>
7  <!ELEMENT dateNaiss (#PCDATA)>
8  <!ELEMENT email (#PCDATA)>
9  <!ELEMENT role (#PCDATA)>
10
11 <!ENTITY copy "(C) ">
12 <!ENTITY licence "Creative Commons CC-BY-SA">
13 <!ENTITY ga "Guillaume Allegre">
```

Remarques :

- ▶ syntaxe spécifique aux DTD
- ▶ ajout d'entités possible dans la DTD

XML "bien défini"

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE annuaire SYSTEM "annuaire.dtd">
3  <annuaire>
4      <entree numero="1">
5          <etatCivile>
6              <nom>Allegre</nom>
7              <prenom>Guillaume</prenom>
8          </etatCivile>
9          <email>allegre.guillaume@free.fr</email>
10         <role>enseignant</role>
11     </entree>
12     <entree numero="2">
13         <etatCivile>
14             <nom>Pittion</nom>
15             <prenom>Sebastien</prenom>
16         </etatCivile>
17         <email>sebastien.pittion@gmail.com</email>
18         <role>enseignant</role>
19     </entree>
```

XML non conforme! (balise etapCivil, numero)

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <!DOCTYPE annuaire SYSTEM "annuaire.dtd">
3 <annuaire>
4     <entree numero="1">
5         <etapCivil>
6             <nom>Allegre</nom>
7             <prenom>Guillaume</prenom>
8         </etapCivil>
9         <email>allegre.guillaume@free.fr</email>
10        <role>enseignant</role>
11    </entree>
12    <entree>
13        <etatCivil>
14            <nom>Pittion</nom>
15            <prenom>Sebastien</prenom>
16        </etatCivil>
17        <email>sebastien.pittion@gmail.com</email>
18        <role>enseignant</role>
19    </entree>
```

xmllint - document non conforme DTD

```
xmllint --noout --valid annuaire2.xml
```

```
annuaire2-err.xml:8: element etapCivil: validity error :  
    No declaration for element etapCivil
```

```
</etapCivil>
```

```
^
```

```
annuaire2-err.xml:11: element entree: validity error :
```

```
    Element entree content does not follow the DTD, expecting
```

```
</entree>
```

```
^
```

```
annuaire2-err.xml:19: element entree: validity error :
```

```
    Element entree does not carry attribute numero
```

```
</entree>
```

```
^
```

Validation par schéma - XSD

Limitations des DTD

- ▶ syntaxe obsolète (SGML)
- ▶ fonctionnalités de contrôle limitées

XSD (XML Schema Definition)

- ▶ syntaxe XML !
- ▶ application des outils XML standard (validation, parcours, transformation)
- ▶ fusion, réutilisation, héritage des schémas

XSD - fichier annuaire.xsd 1/2

```
1  <?xml version="1.0"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4  <xs:element name="annuaire">
5    <xs:complexType>
6      <xs:sequence>
7        <xs:element name="entree" maxOccurs="unbounded">
8          <xs:complexType>
9            <xs:sequence>
10              <xs:element name="etatCivil">
11                <xs:complexType>
12                  <xs:sequence>
13                    <xs:element name="nom" type="xs:string"/>
14                    <xs:element name="prenom" type="xs:string"/>
15                    <xs:element name="dateNais" type="xs:string" minOccurs="1" maxOccurs="1"/>
16                  </xs:sequence>
17                </xs:complexType>
18              </xs:element> <!--etatCivil-->
```


XSD - fichier `annuaire.xsd` 2/2

```
19         <xs:element name="email" type="xs:string"/>
20         <xs:element name="role" type="xs:string"/>
21     </xs:sequence>
22     <xs:attribute name="numero" type="xs:string" use="optional"/>
23 </xs:complexType>
24 </xs:element> <!--entree-->
25 </xs:sequence>
26 </xs:complexType>
27 </xs:element> <!--annuaire-->
28
29 </xs:schema>
```

Remarques

- ▶ utilisation obligatoire du mécanisme d'espace de noms `xmlns`
- ▶ bien plus verbeux que les DTD

Exemple de conformité XSD

```
xmllint --schema annuaire.xsd --valid --noout  
annuaire1.xml
```

```
annuaire2-err.xml:8: element etapCivil: validity error :  
No declaration for element etapCivil  
</etapCivil>
```

```
^  
annuaire2-err.xml:11: element entree: validity error :  
Element entree content does not follow the DTD, expecting (  
</entree>
```

```
^  
annuaire2-err.xml:19: element entree: validity error :  
Element entree does not carry attribute numero  
</entree>
```

```
^
```