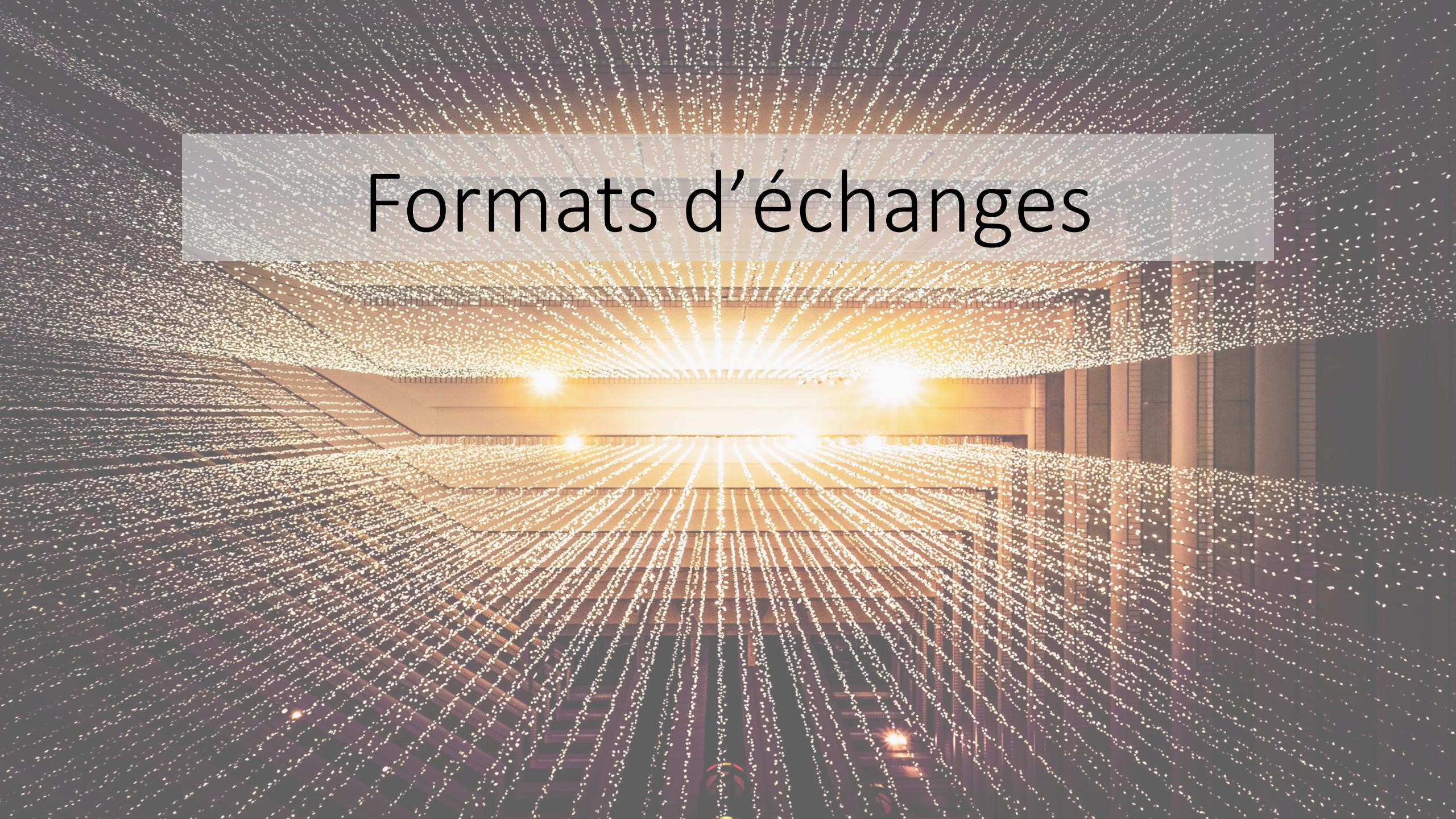


Formats d'échanges



Objectifs

- Comparaison formats texte vs binaire
- Notation XML
- Notation JSON

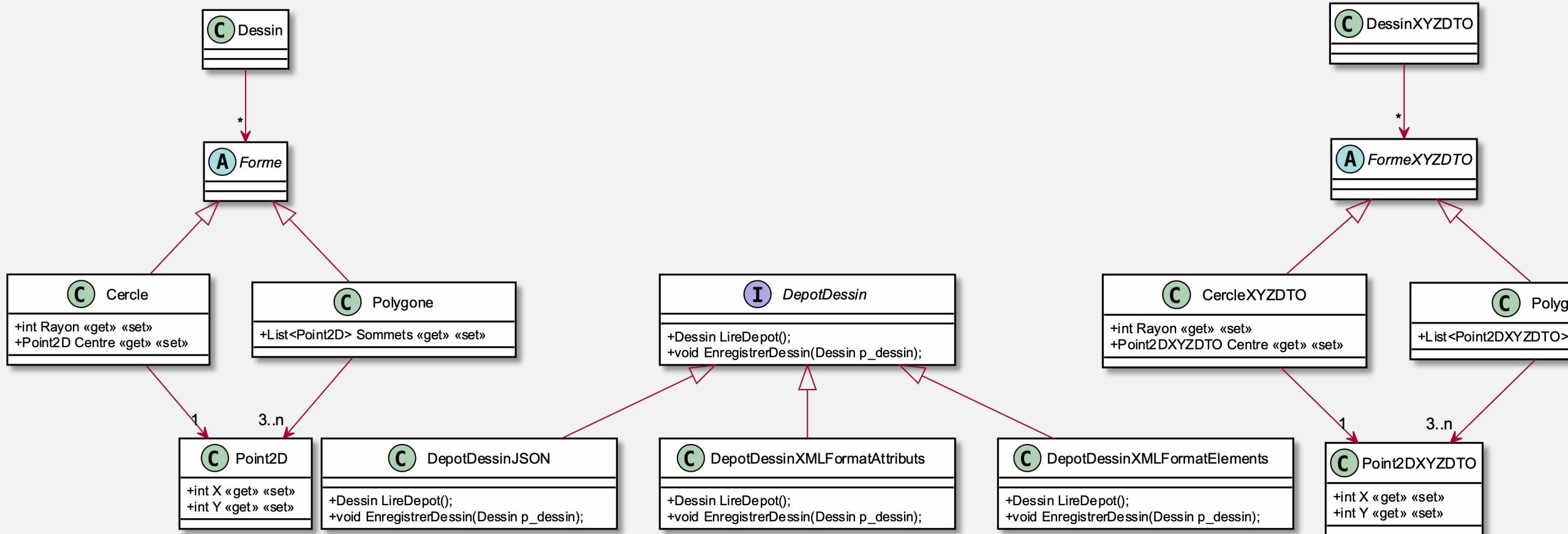
Format texte vs binaire

- **Binaire :**
 - + Rapide (moins d'effort de conversion)
 - + Non volumineux
 - Dépendant de l'architecture des processus pour certains types (Big Endian / Little Endian)
 - Non lisible dans un éditeur classique
- **Utilisation :**
 - Économie d'espace et de bande passante
- **Types :**
 - ASN1
 - Fichiers propriétaires

Formats texte vs binaire

- **Texte :**
 - + Lisible avec un simple éditeur de texte
 - + Dans la majorité des cas, interprétable par directement par un humain
 - Volumineux et verbeux :
 - les valeurs entières sont écrites en texte
 - les formats modernes ont une partie non négligeable du volume est occupé par des éléments descriptifs et non des données
 - Problème d'encodage des caractères accentués et des retours de chariot
- **Utilisation :**
 - Transfert de données sur réseaux (souvent sous forme compressée)
 - Assurer la compatibilité des données inter-systèmes
- **Types :**
 - Fichiers plats : par position, CSV, Wavefront (.obj)
 - XML, JSON, YAML, etc.

Exemple utilisé par la suite



XML

- XML : *eXtensible Markup Language*
- Langage pour structurer de l'information
- N'indique pas comment traiter l'information
- Est très permissif :
 - Quelques règles de base
 - Le reste est au choix du concepteur
 - Est très verbeux
 - Peut coder un seul entier sur une vingtaine de caractères

XML – Syntaxe

- La première ligne contient l'entête XML
- Langage de balises
- Les valeurs peuvent soit être représentées dans un élément soit des attributs

```
<?xml version="1.0" encoding="utf-8"?>
<Dessin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Formes>
    <Forme xsi:type="Polygone">
      <Sommets>
        <Point2D>
          <X>-1</X>
          <Y>-1</Y>
        </Point2D>
        <Point2D>
          <X>1</X>
          <Y>-1</Y>
        </Point2D>
        <Point2D>
          <X>0</X>
          <Y>1</Y>
        </Point2D>
      </Sommets>
    </Forme>
```

```
<?xml version="1.0" encoding="utf-8"?>
<Dessin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Formes>
    <Forme xsi:type="Polygone">
      <Sommets>
        <Point2D X="-1" Y="-1" />
        <Point2D X="1" Y="-1" />
        <Point2D X="0" Y="1" />
      </Sommets>
    </Forme>
    <Forme xsi:type="Polygone">
      <Sommets>
        <Point2D X="0" Y="0" />
        <Point2D X="2" Y="2" />
        <Point2D X="0" Y="4" />
        <Point2D X="-2" Y="-2" />
      </Sommets>
    </Forme>
```

XML – C#

- En C#, nous utiliserons la classe « `XmlSerializer` » qui est native dans le cadiciel
- Les deux principales méthodes sont :
 - `Serialize` : Écrit un objet passé en paramètre sous forme XML
 - `Deserialize` : Lit une chaîne contenant un XML passée en paramètres et le transforme en objet
- Par défaut, la sérialisation crée un élément par propriété. Le nom est le même que la propriété
- On peut modifier le comportement de la sérialisation avec des attributs débutant par « `Xml` »
- Demo

JSON

- JSON : JavaScript Object Notation
- Basé sur JavaScript : tout indiquer pour communiquer avec un client Web
- Format plus léger qu'XML
- Facile à lire et à écrire : des bibliothèques de fonctions / classes dans la plupart des langages
- L'extension de fichiers est « .json »

JSON – Notation

- Notations :
 - Objets : {...}
 - Propriétés : "<nom propriété>" :<valeur>
 - Collections : [...]
(peuvent contenir des objets ou des valeurs)
- Les caractères blancs sont ignorés

```
{  
  "Formes": [  
    {  
      "$type":  
        "Module06_Formats_Echanges_PreparationCours.Depots.JSON.DTO.PolygoneJSONDTO,  
        "Module06_Formats_Echanges_PreparationCours",  
      "Sommets": [  
        {  
          "X": -1,  
          "Y": -1  
        },  
        {  
          "X": 1,  
          "Y": -1  
        },  
        {  
          "X": 0,  
          "Y": 1  
        }  
      ]  
    },  
    {  
      "$type":  
        "Module06_Formats_Echanges_PreparationCours.Depots.JSON.DTO.PolygoneJSONDTO,  
        "Module06_Formats_Echanges_PreparationCours",  
      "Sommets": [  
        {  
          "X": 0,  
          "Y": 0  
        },  
        [...]  
      ]  
    }  
  ]  
}
```

JSON – C#

- En C#, nous installerons le package nuget « Newtonsoft.Json »
- Nous utiliserons principalement les méthodes suivantes de la classe JsonConvert :
 - SerializeObject : Écrit un objet passé en paramètre sous forme JSON
 - DeserializeObject : Lit une chaîne contenant un JSON passée en paramètres et le transforme en objet
- Par défaut, la sérialisation crée un élément par propriété. Le nom est le même que la propriété
- On peut modifier le comportement de la sérialisation avec des attributs débutant par « Json »
- Demo