# Master SESI [M1]
# UE MOBJ – Rattrapage

Jean-Paul CHAPUT

Juin 2016

**Durée : 2 heures – Tous documents autorisés**

**Le barème est donné à titre indicatif et peut être modifié par le correcteur**

**Écrivez lisiblement, un texte difficilement déchiffrable sera toujours considéré comme faux**

## Représentation mémoire d'un arbre XML

Un document XML est représentable sous la forme d'un arbre. Chaque paire de balises (`tag`) ouvrante et fermante (comme `<packagelist>` et `</packagelist>`) est associée à un noeud. Intégrée à la balise ouvrante on peut trouver une ou plusieurs propriétés (comme dans `<packagereq type="mandatory">`
Le contenu inclus entre les balises ouvrantes et fermantes peut être, soit du texte simple (comme `<name>XFce</name>`) qui sera rangé dans l'attribut `_contents` du noeud, soit d'autres paires de balises qui constitueront un ensemble de noeud fils (childs). Un exemple de la structure arborescente est donné figure 3.

Considérant l'arbre que nous voulons représenter, nous avons besoin de trois types de noeuds : (arbre d'héritage demandé : figure 1)

— Les noeuds sans fils et sans propriétés.
  Classe `XmlNode`. Examples : `id`, `default`.
— Les noeuds sans fils et un nombre quelconque de propriétés.
  Classe `XmlNodeProps`. Examples : `name`, `packagereq`.
— Les noeuds avec plusieurs fils et sans propriétés.
  Classe `XmlNodeChilds`. Examples : `group`, `packagelist`.

La classe `XmlNode` peut-elle être abstraite ?

On créera aussi une classe indépendante (sans lien de parenté avec `XmlNode`) `Property` pour stocker une propriété.

**Remarque :** dans toute la suite, pour gérer les textes, on utilisera des `string` et on ne se préocupera pas de leur allocation/désallocation.

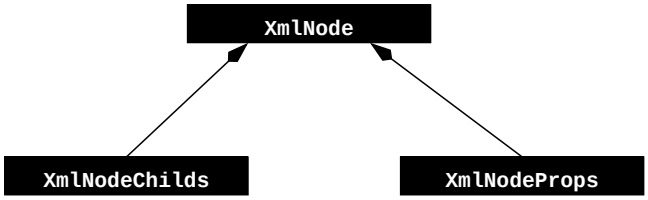


FIGURE 1 – Relations d'héritage des classes

Exemple de document XML :

```
<comps>
  <group>
    <id>core</id>
    <uservisible>false</uservisible>
    <default>true</default>
    <name>Core Components</name>
    <packagelist>
      <packagereq type="mandatory">kernel</packagereq>
      <packagereq type="mandatory">glibc</packagereq>
      <packagereq type="mandatory">bash</packagereq>
    </packagelist>
  </group>

  <!-- SL LIP6/SoC Addition -->
  <group>
    <id>xfce-desktop</id>
    <uservisible>true</uservisible>
    <default>false</default>
    <name>XFce</name>
    <name lang="fr">XFce</name>
    <description>A Lightweight desktop environment for *NIX.</description>
    <description lang="fr">Un environnement graphique léger pour *NIX.</description>
    <packagelist>
      <packagereq type="mandatory">xfce-mcs-manager</packagereq>
      <packagereq type="mandatory">xfce-mcs-plugins</packagereq>
      <packagereq type="mandatory">xfce4-panel</packagereq>
      <packagereq type="mandatory">xfce4-session</packagereq>
      <packagereq type="mandatory">xfwm4</packagereq>
      <packagereq type="default">xfce4-weather-plugin</packagereq>
      <packagereq type="default">xfce4-datetime-plugin</packagereq>
      <packagereq type="default">xfce4-mount-plugin</packagereq>
      <packagereq type="default">xfce4-systemload-plugin</packagereq>
      <packagereq type="default">xfce4-netload-plugin</packagereq>
      <packagereq type="optional">xfce4-mpc-plugin</packagereq>
    </packagelist>
  </group>
</comps>
```

**Important :** l'exemple de fichier XML donné çi-dessus présente l'ensemble des types de noeuds que l'on se propose de gérer. Il n'est pas demandé de prévoir d'autres cas ou des cas plus généraux.

### Fonctions nécéssaires à la construction de l'arbre

Les constructeurs des différentes classes de noeuds prendront un et un seul argument obligatoire : le nom de la balise.

Une fois créé, on remplira un noeud à l'aide des fonctions membres de modification suivantes :
— `void setContents(char*)`, qui positionne l'attribut `_contents`.
— `Property* addProperty(char*,char*)`, qui ajoute une propriété (elle prendra deux arguments : le nom de la propriété et sa valeur).
— `XmlNode* addChild(char* tag)`, qui ajoute un fils. Elle prendra en unique argument une balise et renverra un pointeur sur l'objet nouvellement crée `XmlNode`.

**Usine à noeud :** pour permettre une allocation plus aisée des différentes classes de noeuds, implanter une fonction de création générique. Cette fonction prendra une chaîne de caractères en argument donnant la balise associée au noeud à créer et renverra un noeud du type correspondant à cette balise. Afin de *ranger proprement* les fonctions, ce sera une méthode statique de la classe de base `XmlNode`. Prototype de la fonction de création :

```
XmlNode* XmlNode::Create(char* tag)
```

**Remarque :** pour que l'usine à noeud puisse fonctionner il faut qu'il existe une relation entre le nom du `tag` et le type du noeud, ce qui est bien le cas. Par example, `comps`, `group` & `packagelist` sont les trois noeuds de type `XmlNodeChilds`.

### Accès aux propriétés

Créer une fonction d'accès par nom, qui retourne une propriété en fonction de son nom :

```
Property* getProperty(char*)
```

**<packagereq type="mandatory"> xfwm4 </packagereq>**

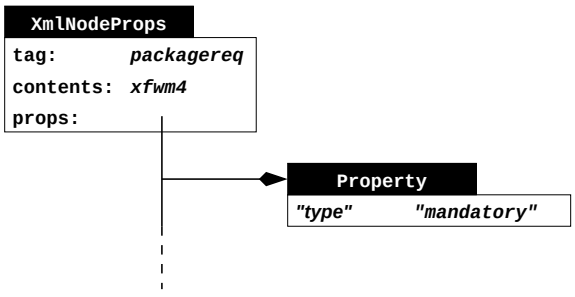


FIGURE 2 – Exemple de noeud simple XML

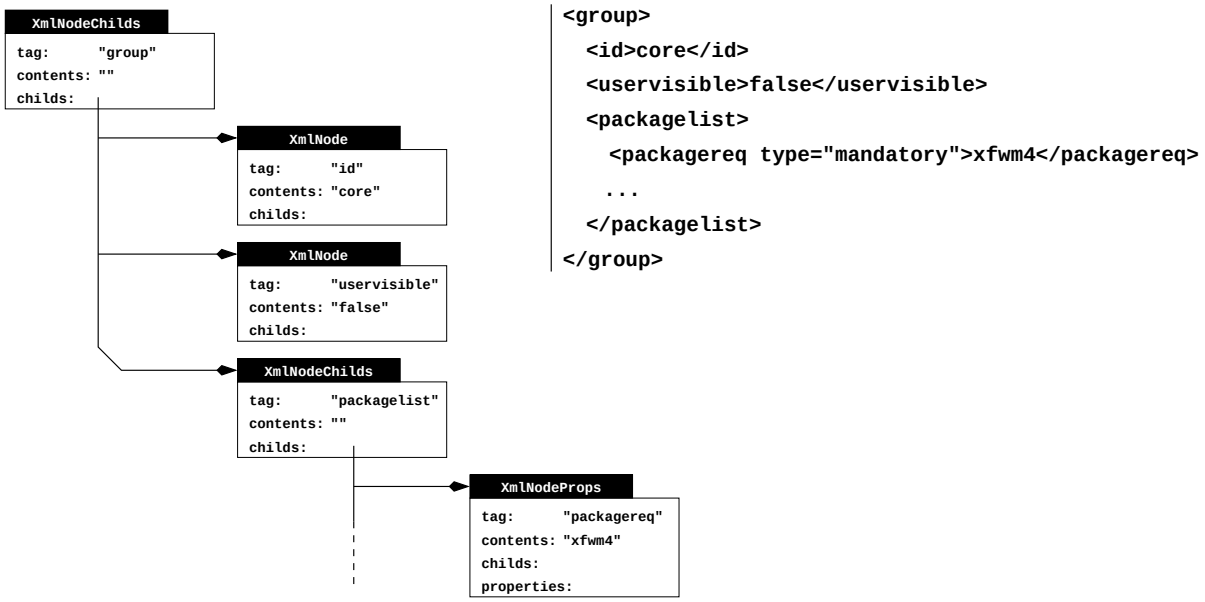


FIGURE 3 – Exemple d'arbre XML

### Fonctions nécéssaires à l'affichage l'arbre

On désire réaliser une fonction membre d'affichage qui restitue le formatage initial de l'arbre XML. Cette fonction sera récursive, c'est à dire que l'affichage d'un noeud entraine celui de ses éventuels fils. Pour un affichage plus clair il faut générer le niveau d'indentation. Prototype de la fonction d'affichage :

```
void _print(ostream& o, int indentLevel)
```

Enfin, ajouter le mécanisme nécéssaire pour pouvoir afficher directement dans un flux.

### Travail demandé

Proposer une implantation complète pour les classes `XmlNode`, `XmlNodeProps`, `XmlNodeChilds` & `Property`.

**Question 1** Implantation de `XmlNode` **6pt**

**Question 2** Implantation de `XmlNodeProps` **5pt**

**Question 3** Implantation de `XmlNodeChilds` **6pt**

**Question 4** Implantation de `Property` **3pt**

# Corrigé Rattrapage UE MOBJ – Juin 2016

## Réponse à la question 1 **6pt**

**Barème :** Constructeurs & transmission des args ..... **1pt**
Attributs dans les bonnes classes ..... **0.5pt**
`XmlNode` n'est pas abstraite ..... **0.5pt**
`addChild()`, `addProperty()` virtuelles ..... **0.5pt**
`addChild()` vide dans `XmlNode` ..... **0.5pt**
Accès aux propriétes (virtuelle) ..... **0.5pt**
Accès aux attributs (non-virtuelles) ..... **0.5pt**
Implantation de l'usine à noeud ..... **1pt**
Fonction d'affichage `_print()` ..... **0.5pt**
Fonction d'affichage dans un flux ..... **0.5pt**

Dans la correction, il ne sera pas tenu compte de la distinction entre `string`, `const char*` et `char*`. De même on ne tiendra pas compte des méthodes *approximatives* utilisées pour comparer deux chaînes de caractères.

```
class XmlNode {
  private:
    string  _tag;
    string  _contents;
  public:
                      XmlNode     ( string tag ) : _tag(tag), _contents() {};
    virtual           ~XmlNode    ();
  public:
            void      setContents ( string contents ) { _contents = contents; }
    virtual Property* addProperty ( string name
                                  , string value ) { return NULL; }
    virtual XmlNode*  addChild    ( string tag   );
    virtual Property* getProperty ( string name  );
    inline  string    getTag      () const { return _tag; }
    inline  string    getContents () const { return _contents; }
            void      _indent     ( ostream& o, int indentLevel ) const;
    virtual void      _print      ( ostream& o, int indentLevel ) const;
    static  XmlNode*  create      ( string tag   );
    friend  ostream&  operator<<  ( ostream& o, const XmlNode& n );
};

          XmlNode::~XmlNode    () {}
XmlNode*  XmlNode::addChild    ( string tag ) { return NULL; }
Property* XmlNode::addProperty ( string name, string value ) { return NULL; }
Property* XmlNode::getProperty ( string name ) { return NULL; }

void XmlNode::_indent ( ostream& o, int indentLevel ) const
{ for ( int i=0 ; i<indentLevel ; i++ ) o << "␣␣"; }

void XmlNode::_print ( ostream& o, int indentLevel ) const
{
  _indent( o, indentLevel );
  o << "<" << _tag << ">" << _contents << "</" << _tag << ">" << "\n";
}
```

```
ostream& operator<< ( ostream& o, const XmlNode& n )
{ n._print( o, 0 ); return o; }

XmlNode* XmlNode::create ( string  tag )
{
  if (    ( stag == "id"          )
       || ( stag == "default"     )
       || ( stag == "uservisible" ) )
    return new XmlNode ( tag );

  if (    ( stag == "name"        )
       || ( stag == "packagereq"  )
       || ( stag == "description" ) )
    return new XmlNodeProps ( tag );

  if (    ( stag == "comps"       )
       || ( stag == "group"       )
       || ( stag == "packagelist" ) )
    return new XmlNodeChilds ( tag );

  cerr << "[ERROR] Unsupported node tag: " << tag << endl;

  return NULL;
}
```

## Réponse à la question 2 **5pt**

**Barème :** Implantation de l'héritage ..... **1pt**
Attribut et type de `_properties` ..... **1pt**
Implantation du destructeur (virtuel) ..... **1pt**
Implantation des get/set property ..... **1pt**
Fonction d'affichage dans un flux ..... **1pt**

```
class XmlNodeProps : public XmlNode {
  private:
    vector<Property*> _properties;
  public:
                      XmlNodeProps  ( string tag ) : XmlNode(tag) {};
    virtual           ~XmlNodeProps ();
  public:
    virtual Property* addProperty   ( string name
                                    , string value );
    virtual Property* getProperty   ( string name  );
    virtual void      _print        ( ostream& o, int indentLevel ) const;
};

XmlNodeProps::~XmlNodeProps ()
{ for ( size_t i=0 ; i<_properties.size() ; i++ ) delete _properties[i]; }
```

```
Property* XmlNodeProps::addProperty ( string name, string  value )
{
  Property* property = getProperty( name );
  if (property) {
    cerr << "[ERROR] Duplicated property: " << name << "\n" << endl;
    return property;
  }
  property = new Property ( name, value );
  _properties.push_back( property );
  return property;
}

Property* XmlNodeProps::getProperty ( string name )
{
  for ( size_t i=0 ; i<_properties.size() ; i++ )
    if ( string(_properties[i]->getName()) == string(name) )
      return _properties[i];
  return NULL;
}

void XmlNodeProps::_print ( ostream& o, int indentLevel ) const
{
  _indent( o, indentLevel );
  o << "<" << _tag;
  for ( size_t i=0 ; i<_properties.size() ; i++ ) {
    o << " "   << _properties[i]->getName()
      << "=\"" << _properties[i]->getValue() << "\"";
  }
  o << ">" << _contents << "</" << _tag << ">" << "\n";
}
```

## Réponse à la question 3 **6pt**

**Barème :** Implantation de l'héritage ..... **1pt**
Attributs corrects ..... **1pt**
Destructeur virtuel & implantation ..... **1pt**
Implantation de `addChild()` ..... **2pt**
Fonction d'affichage dans un flux ..... **1pt**

```
class XmlNodeChilds : public XmlNode {
  private:
    vector<XmlNode*> _childs;
  public:
                     XmlNodeChilds  ( string  tag ) : XmlNode(tag) {};
    virtual          ~XmlNodeChilds ();
  public:
    virtual XmlNode* addChild       ( string  tag );
    virtual void     _print         ( ostream& o, int indentLevel ) const;
};
```

```
XmlNodeChilds::~XmlNodeChilds ()
{ for ( size_t i=0 ; i<_childs.size() ; i++ ) delete _childs[i]; }


XmlNode* XmlNodeChilds::addChild ( string  tag )
{
  XmlNode* node = create( tag );
  _childs.push_back( node );
  return node;
}


void XmlNodeChilds::_print ( ostream& o, int indentLevel ) const
{
  _indent( o, indentLevel );
  o << "<" << _tag << ">";
  if (not _contents.empty()) o << _contents;
  o << "\n";

  for ( size_t i=0 ; i<_childs.size() ; i++ )
    _childs[i]->_print( o, indentLevel + 1 );

  _indent( o, indentLevel );
  o << "</" << _tag << ">" << "\n";
}
```

## Réponse à la question 4 **3pt**

**Barème :** Pas d'héritage ..... **0.5pt**
Attributs corrects ..... **1pt**
Constructeur ..... **0.5pt**
Destructeur par défaut suffisant ..... **0.5pt**
Accesseurs ..... **0.5pt**

```
class Property {
  private:
    string  _name;
    string  _value;
  public:
    Property ( string name, string value )
             : _name(name), _value(value)
    {}
  public:
    string  getName  () const { return _name; }
    string  getValue () const { return _value; }
};
```

Et en prime, un petit programme de test...

```
int main ( int argc, char* argv[] )
{
  XmlNode* root = XmlNode::create ( "comps" );

  XmlNode* group       = root->addChild  ( "group" );
  XmlNode* id          = group->addChild ( "id" );
  XmlNode* visible     = group->addChild ( "uservisible" );
  XmlNode* ndefault    = group->addChild ( "default" );
  XmlNode* name        = group->addChild ( "name" );
  XmlNode* packagelist = group->addChild ( "packagelist" );

  id->setContents       ( "core" );
  visible->setContents  ( "false" );
  ndefault->setContents ( "true" );
  name->setContents     ( "Core␣Components" );

  XmlNode* packagereq = packagelist->addChild ( "packagereq" );
  packagereq->setContents ( "kernel" );
  packagereq->addProperty ( "type", "mandatory" );

  packagereq = packagelist->addChild ( "packagereq" );
  packagereq->setContents ( "glibc" );
  packagereq->addProperty ( "type", "mandatory" );

  packagereq = packagelist->addChild ( "packagereq" );
  packagereq->setContents ( "bash" );
  packagereq->addProperty ( "type", "mandatory" );

  cout << "XML␣tree:" << endl;
  cout << *root << endl;

  delete root;

  exit ( 0 );
}
```