

# Rapport sur la mise en place du parse XML avec Python afin d'utiliser un fichier de configuration XML

Alexis Carreau  
*L2 Informatique*

```
sh-4.1$ hadoop dfs -cat /user/hduser/inputxmlweather/weatherall.xml | more
<?xml version="1.0"?>
<WeatherResponse>
  <WeatherDetails>
    <Zipcode>60657</Zipcode>
    <State>IL</State>
    <City>Chicago</City>
    <Forecast>
      <ForecastDate>2012-04-18</ForecastDate>
      <DescriptiveForecast GeneralDesc='Partly Cloudy'>
        <DayTimeDesc>Partly cloudy. High of 68F. Winds from the NNE at 10 to 15 mph.</DayTimeDesc>
        <NightTimeDesc>Partly cloudy. Fog overnight. Low of 25F. Winds from the NNE at 10 to 15 mph.</NightTimeDesc>
      </DescriptiveForecast>
      <Temperature>
        <TempLow>25</TempLow>
        <TempHigh>68</TempHigh>
      </Temperature>
      <Precipitation>
        <PrecipNighttime/>
        <PrecipDaytime>30</PrecipDaytime>
      </Precipitation>
      <SunriseTime>5:30</SunriseTime>
      <SunsetTime>20:03</SunsetTime>
    </Forecast>
  </WeatherDetails>
</WeatherResponse>

<?xml version="1.0"?>
<WeatherResponse>
  <WeatherDetails>
    <Zipcode>60657</Zipcode>
    <State>IL</State>
    <City>Chicago</City>
    <Forecast>
      <ForecastDate>2012-04-19</ForecastDate>
      <DescriptiveForecast GeneralDesc='Clear Skies'>
```

FIGURE 1 – Exemple de fichier XML

# Table des matières

<b>1</b>	<b>La nécessité du XML</b>	<b>2</b>
<b>2</b>	<b>XML et Python : LXML</b>	<b>2</b>
<b>3</b>	<b>Le module XML et le fichier de configuration</b>	<b>3</b>
<b>4</b>	<b>L'application dans notre IDE</b>	<b>4</b>
4.1	Les thèmes . . . . .	4
4.2	L'assistance vocale . . . . .	5
<b>5</b>	<b>Conclusion</b>	<b>6</b>

# 1 La nécessité du XML

Au fur et à mesure de l'ajout de fonctionnalités dans notre IDE, l'utilisation du XML s'est imposé du fait de son efficacité et également par convention. Nous devons ainsi lire, parcourir et écrire dans un fichier XML.

## 2 XML et Python : LXML

Pour ce faire, nous avons utilisé LXML, qui nous permet d'utiliser des outils de parse xml dans notre IDE avec Python. En effet, `etree`, après son import, nous permet de parser un fichier XML. Nous pouvons le voir sur l'image ci dessous, par rapport à l'import. Sur cette dernière, nous voyons également le parcours d'un fichier XML en parcourant ses différentes balises :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from lxml import etree

tree = etree.parse("data.xml")
for user in tree.xpath("/users/user/nom"):
    print(user.text)
```

FIGURE 2 – Import de LXML et parcours XML

Nous avons de plus, beaucoup d'outils disponibles pour parcourir et modifier un fichier XML :

<code>addnext(element)</code>	: Ajoute un élément en tant que frère juste après l'élément
<code>addprevious(element)</code>	: Ajoute un élément en tant que frère juste avant l'élément
<code>append(element)</code>	: Ajoute un sous-élément à la fin de l'élément
<code>clear()</code>	: Supprime tous les sous-éléments
<code>extends(elements)</code>	: Etend les éléments passé en paramètre
<code>find(path)</code>	: Recherche le premier sous-élément qui correspond au tag/path
<code>findall(path)</code>	: Recherche tous les sous-éléments qui correspondent au tag/path
<code>findtext(path)</code>	: Trouve le texte du premier sous-élément qui correspond au tag/path
<code>get(key)</code>	: Recupère l'attribut d'un élément
<code>getchildren()</code>	: Retourne tous les enfants d'un élément ! attention déprécié pour <code>list(element)</code>
<code>getnext()</code>	: Retourne le prochain élément frère
<code>getparent()</code>	: Retourne le parent de l'élément
<code>getprevious()</code>	: Retourne l'élément frère précédant
<code>index(child)</code>	: Trouve la position de l'élément
<code>insert(index)</code>	: Insère un sous-élément à la position indiquée
<code>items()</code>	: Retourne les attributs d'un élément (dans un ordre aléatoire)
<code>keys()</code>	: Retourne une liste des noms des attributs
<code>remove(element)</code>	: Supprime l'élément passé en paramètre
<code>replace(el1, el2)</code>	: Remplace <code>el1</code> par <code>el2</code>
<code>set(key, value)</code>	: Créer un attribut avec une valeur
<code>values()</code>	: Retourne les valeurs des attributs
<code>xpath(path)</code>	: Evalue une expression xpath

FIGURE 3 – Outils LXML disponibles

Nous avons donc ainsi tous les outils nécessaires pour utiliser le XML dans notre IDE.

### 3 Le module XML et le fichier de configuration

Nous avons donc ainsi créer un module XML pour les fonctions générales de parcours et d'écriture de fichier XML et bien entendu, le fichier de configuration XML. Voici d'ailleurs leur disposition dans notre IDE, ils sont à la racine du projet :

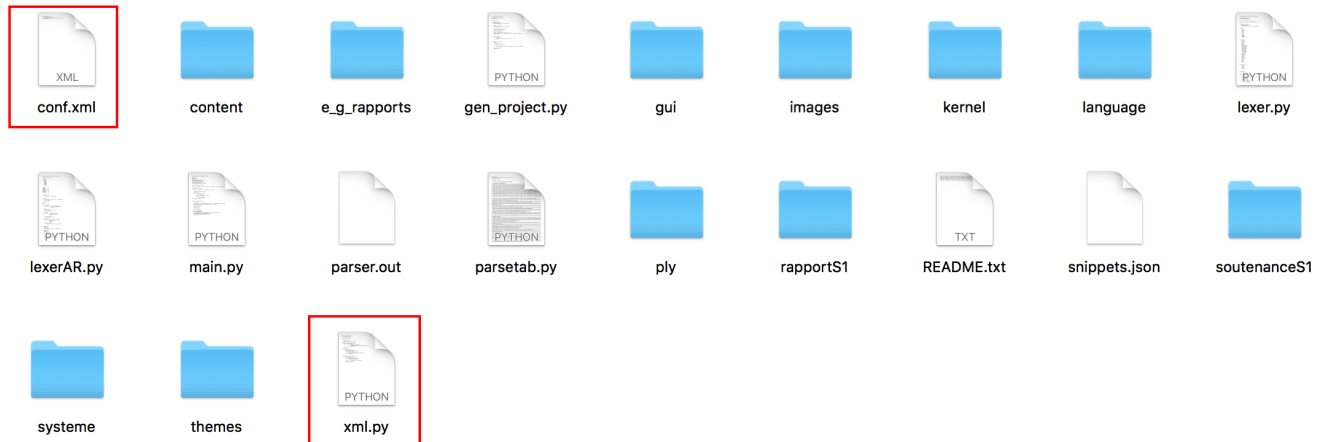


FIGURE 4 – Disposition du module et du fichier de configuration

Dans le module XML, nous avons une fonction `open_xml`, qui ouvre, parse et parcourt le fichier de configuration XML pour retourner un dictionnaire avec comme clés le nom des balises XML, et comme valeurs la valeur des balises. Nous avons également une fonction `write_xml` qui permet d'écrire des modifications dans le fichier de configuration. On rentre le nom d'une balise et la valeur voulue en arguments, la fonction trouve la balise et lui associe cette valeur, puis nous écrivons dans le fichier. Enfin, pour éviter tout problème, si le fichier de configuration n'existe pas, on le crée et lui ajoute les valeurs de configuration de base.

```
xml.py
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  from lxml import etree
5
6  try :
7      open("conf.xml")
8      #tree = etree.parse("conf.xml")
9  except :
10     new = open("conf.xml", "w")
11     new.write("<configuration>\n"
12             "    <theme>basic</theme>\n"
13             "    <assistance_vocale>False</assistance_vocale>\n"
14             "</configuration>")
15     new.close()
16
17 def open_xml():
18     configuration = {}
19     tree = etree.parse("conf.xml")
20     root = tree.getroot()
21     for child in root.getchildren():
22         configuration[child.tag] = child.text
23     return configuration
24
25
26 def write_xml(config, value):
27     tree = etree.parse("conf.xml")
28     root = tree.getroot()
29     for child in root.getchildren():
30         if child.tag == config:
31             mode = root.find(config)
32             mode.text = value
33     tree.write("conf.xml")
```

FIGURE 5 – Module XML

Dans le fichier de configuration XML, nous utilisons une balise pour stocker, le thème sélectionné actuellement, ainsi que le booléen True / False suivant l'activation ou non de l'assistance vocale dans l'interface graphique.

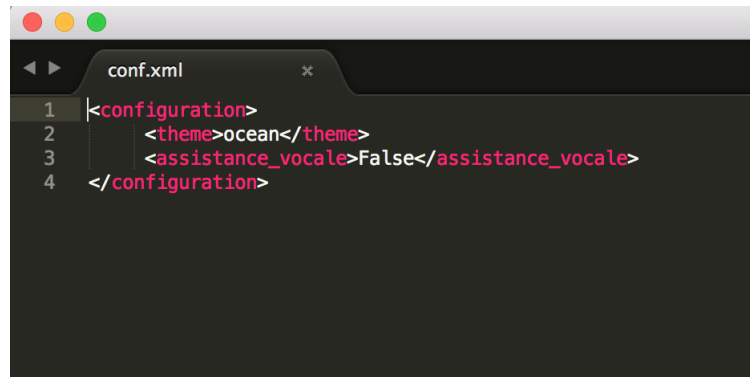


FIGURE 6 – Fichier de configuration

## 4 L'application dans notre IDE

Après avoir défini les différentes fonctions et stockage de variables nécessaires, il ne restait plus qu'à implémenter le XML dans notre IDE. Nous l'avons pour le moment implémentés pour les thèmes et l'assistance vocale.

### 4.1 Les thèmes

Pour implémenter le XML pour les thèmes nous devons donc adapter les différentes fonctions déjà définies. Nous l'avons donc fait dans le module themes :

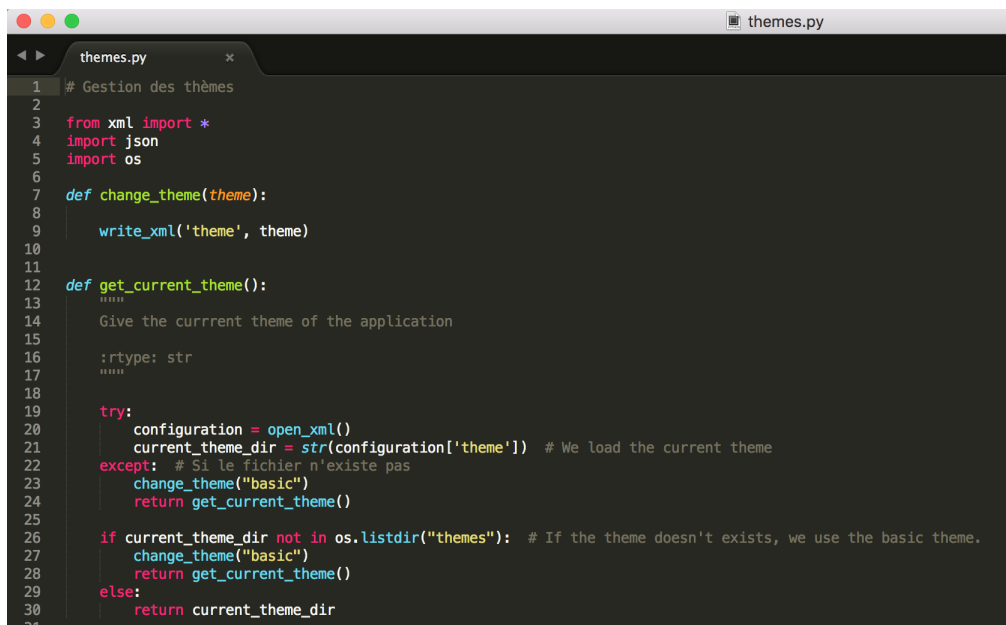


FIGURE 7 – Fonctions pour changer et récupérer le thème courant

Grâce au dictionnaire, nous pouvons ainsi récupérer la valeur du thème actuel en écrivant `configuration['theme']`. Pour modifier le thème, nous utilisons la fonction `write_xml` afin d'associer la valeur en argument à la valeur de la balise `theme` dans le fichier XML.

Nous pouvons voir que la mise à jour du fichier XML s'effectue bien suivant le thème sélectionné du fait de sa mémorisation à l'ouverture et fermeture de l'IDE, voilà l'interface graphique de la sélection du thème :

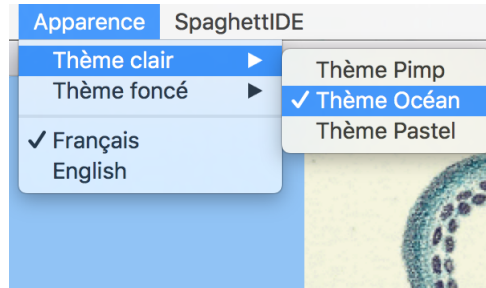


FIGURE 8 – Interface graphique de la sélection du thème

## 4.2 L'assistance vocale

Pour implémenter le XML pour l'assistance vocale nous devons donc encore une fois adapter les différentes fonctions déjà définies. Nous l'avons donc d'abord fait dans le module GUI :

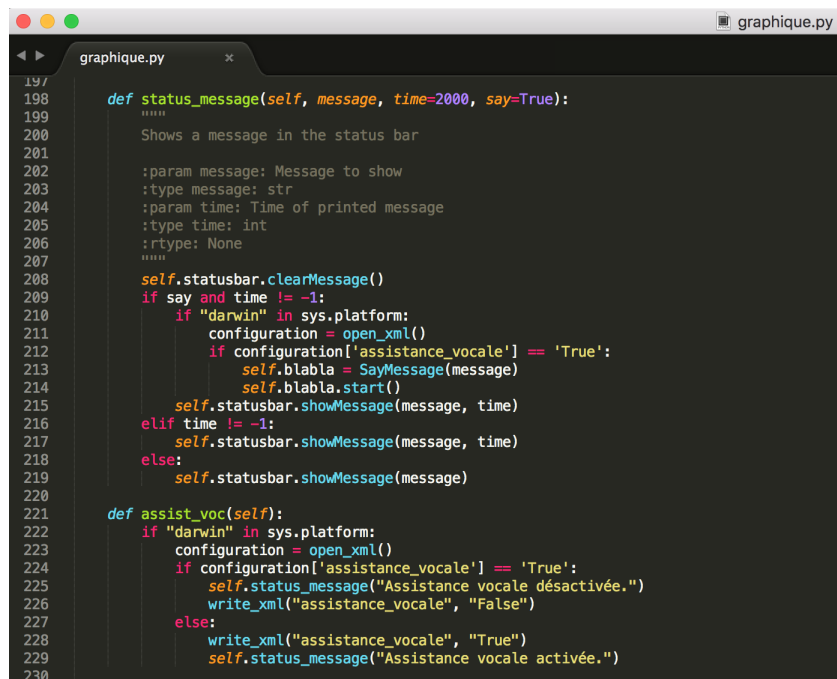
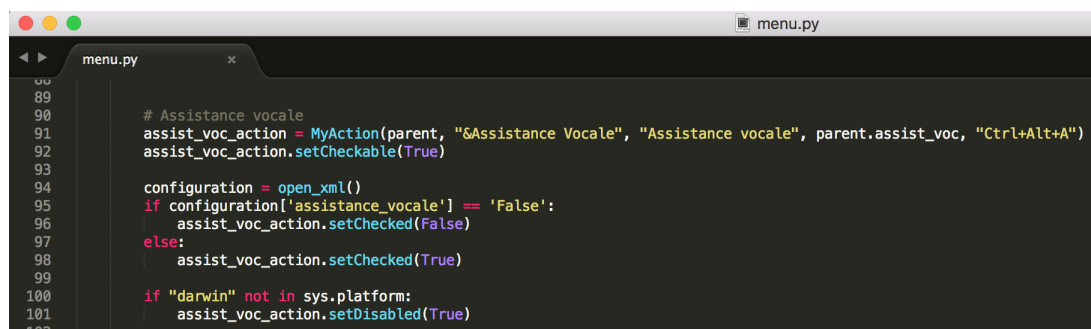


FIGURE 9 – L'assistance vocale et le module GUI

Grâce au dictionnaire, nous pouvons ainsi récupérer la valeur de l'activation de l'assistance vocale actuelle en écrivant `configuration['assistance_vocale']`. Nous pouvons ainsi faire des conditions comme nous pouvons le voir dans l'image précédente. Nous utilisons encore une fois la fonction `write_xml` afin d'associer la valeur en argument à la valeur de la balise `assistance_vocale` dans le fichier XML.

Nous avons enfin adapté les différentes fonctions déjà définies dans le module MENU :



```
89
90 # Assistance vocale
91 assist_voc_action = MyAction(parent, "&Assistance Vocale", "Assistance vocale", parent.assist_voc, "Ctrl+Alt+A")
92 assist_voc_action.setCheckable(True)
93
94 configuration = open_xml()
95 if configuration['assistance_vocale'] == 'False':
96     assist_voc_action.setChecked(False)
97 else:
98     assist_voc_action.setChecked(True)
99
100 if "darwin" not in sys.platform:
101     assist_voc_action.setDisabled(True)
102
```

FIGURE 10 – L'assistance vocale et le module MENU

Avec encore une fois des conditions, nous avons fait en sorte que si l'assistance vocale était désactivée alors elle n'est pas cochée dans le menu fichier et inversement.

Nous pouvons voir que la mise à jour du fichier XML s'effectue bien suivant l'activation ou la désactivation de l'assistance vocale du fait de sa mémorisation à l'ouverture et fermeture de l'IDE, voilà l'interface graphique de la sélection de l'assistance vocale :

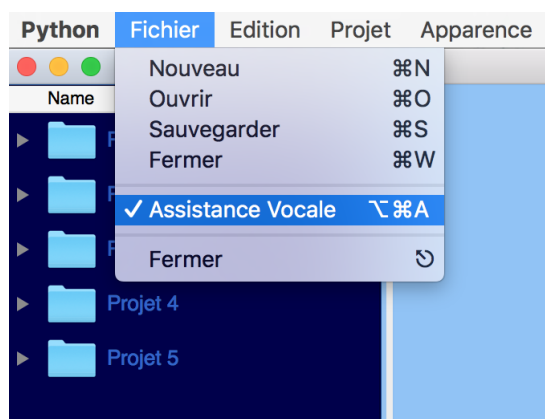


FIGURE 11 – Interface graphique de l'activation ou non de l'assistance vocale

## 5 Conclusion

En conclusion, nous pouvons dire que le XML est vraiment très pratique et nous comptons l'utiliser dans d'autres cas pour notre IDE, par exemple, dès que l'on aura d'autres configurations à ajouter, donc dans le fichier de configuration.