Année universitaire 2016-2017
Université de Caen Basse-Normandie

Rapport sur la mise en place de l'XML avec Python dans notre IDE

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

1 La nécessité du XML

Au fur et à mesure de l'ajout de fonctionnalités dans notre IDE, l'utilisation du XML s'est vite imposée du fait de son efficacité et également par convention. Nous devions ainsi lire, parcourir et écrire dans un fichier XML afin de subvenir à nos besoins dans l'élaboration des différentes fonctionnalités de notre IDE dans différentes conditions. Nous utilisons plusieurs fichiers xml, un pour chaque projet, un pour tous les projets et un fichier de configuration.

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, l'outil "etree", après son import, nous permet de parser un fichier XML et de le parcourir et modifier par la suite.

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

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

FIGURE 2 – Outils LXML disponibles

Nous avions donc ainsi tous les outils nécessaires pour utiliser le XML dans notre IDE. Nous avons d'ailleurs utilisé les outils "find", "getchildren" et "xpath" par exemple pour le parcours du fichier XML.

3 Le module XML et le fichier de configuration

Nous avons donc ainsi créer un module XML pour les fonctions 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.

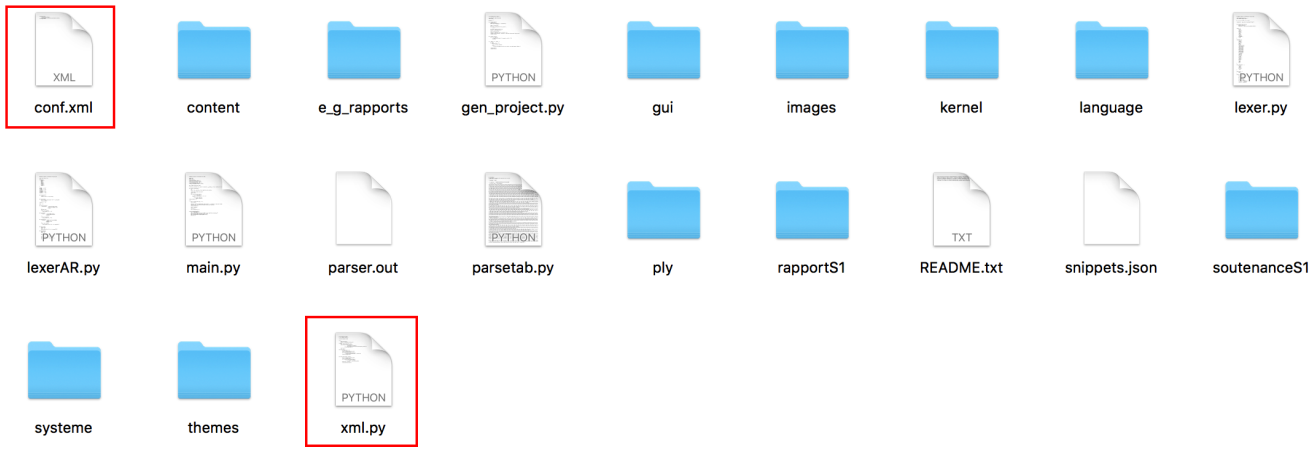

FIGURE 3 – Disposition du module et du fichier de configuration

Dans le module XML, nous avons une fonction open_xml, qui ouvre, parse et parcours 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 ainsi que le fichier comportant tous les projets, on les créé et leur ajoute les valeurs de configuration de base.

Dans le fichier de configuration XML, nous utilisons une balise pour stocker, le thème sélectionné actuellement, le booléen suivant l'activation ou non de l'assistance vocale dans l'interface graphique, le booléen suivant l'activation ou non de l'écran de chargement au démarrage, celui de la numérotation des lignes, le langage de l'IDE (Français ou Anglais) ainsi que la localisation du workplace.

```
<configuration>
    <theme>ocean</theme>
    <assistance_vocale>False</assistance_vocale>
    <loading>False</loading>
    <numerote_lines>True</numerote_lines>
    <language>en</language>
    <current_workplace>/Users/alexiscarreau/workplace/</current_workplace>
</configuration>
```

FIGURE 4 – 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 implémentés pour différentes fonctions et nous allons voir des exemples concrets de leur application.

4.1 Les thèmes

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

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 :

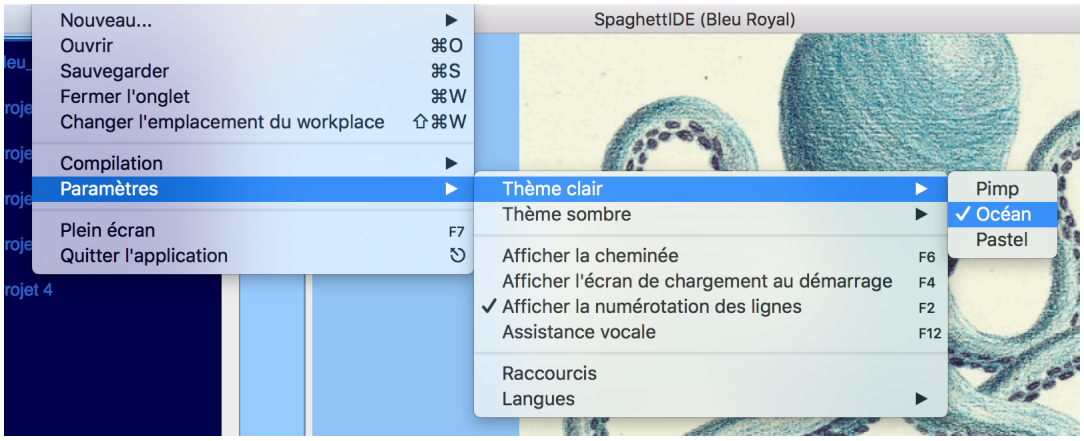

FIGURE 5 – Interface graphique de la sélection du thème

4.2 L'assistance vocale

L'assistance vocale lit seulement les messages affichés dans la barre de statut et permet ainsi de les dire à voix haute en utilisant la fonctionnalité say de MAC OSX. Donc en fait, on récupère le message de la barre de statut et on le fait dire à voix haute avec cette commande : "os.system("say message")" et le message est renouvelé à chaque clic (si l'assistance vocale est activée bien sûr) sur par exemple l'apparence de l'IDE, donc en cliquant sur un thème dans la barre de menu. Pour implémenter le XML pour l'assistance vocale nous devions donc encore une fois adapter les différentes fonctions déjà définies. Nous l'avons donc d'abord fait dans 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. 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 adapté les différentes fonctions déjà définies dans 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 :

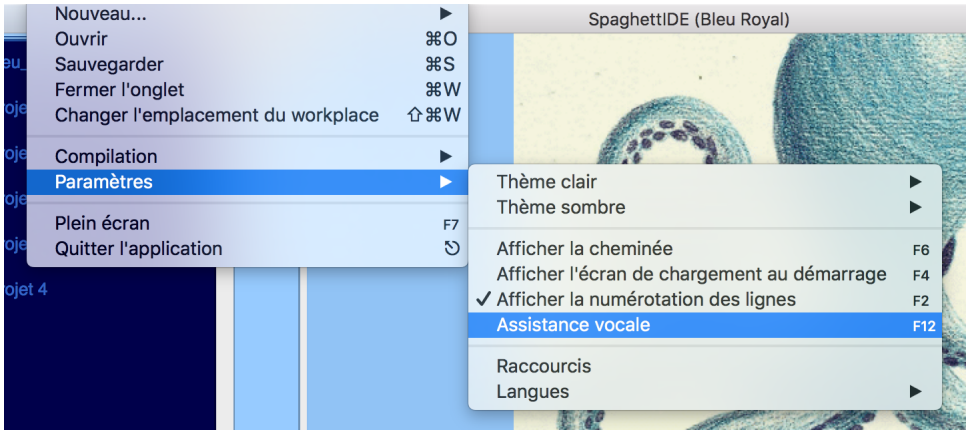

FIGURE 6 – Interface graphique de la sélection de l'assistance vocale

L'assistance vocale n'est disponible pour le moment que sous MAC OSX.

4.3 L'écran de chargement au démarrage

L'écran de chargement au démarrage est sélectionnable via le menu et il sera ainsi affiché à chaque démarrage de l'IDE en premier plan. Pour ce faire, c'est le même principe que précédemment avec le booléan stocké dans le fichier de configuration xml qui est lu à chaque démarrage de l'IDE.

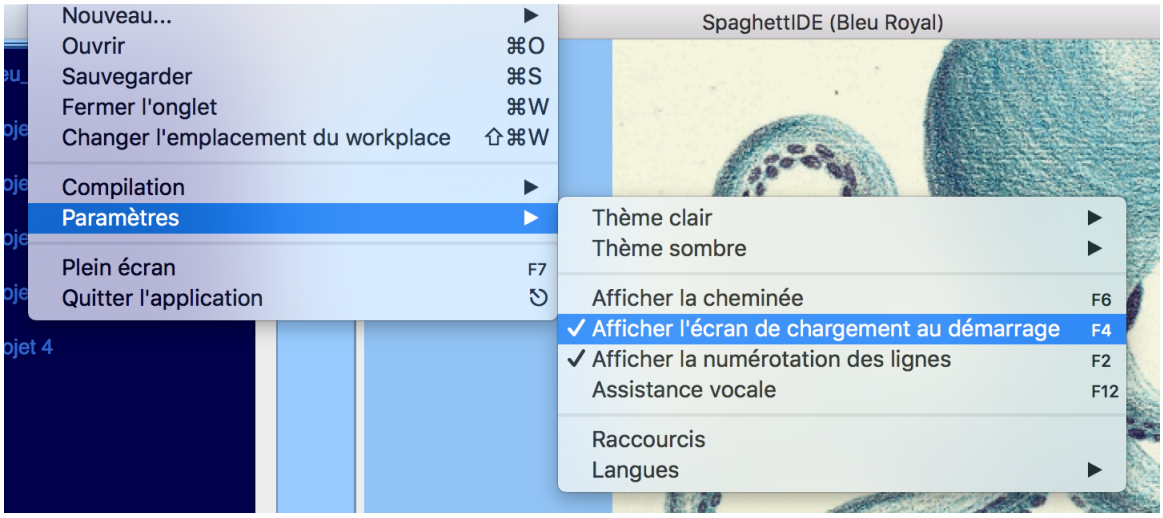

FIGURE 7 – Interface graphique de la sélection de l'écran de chargement au démarrage

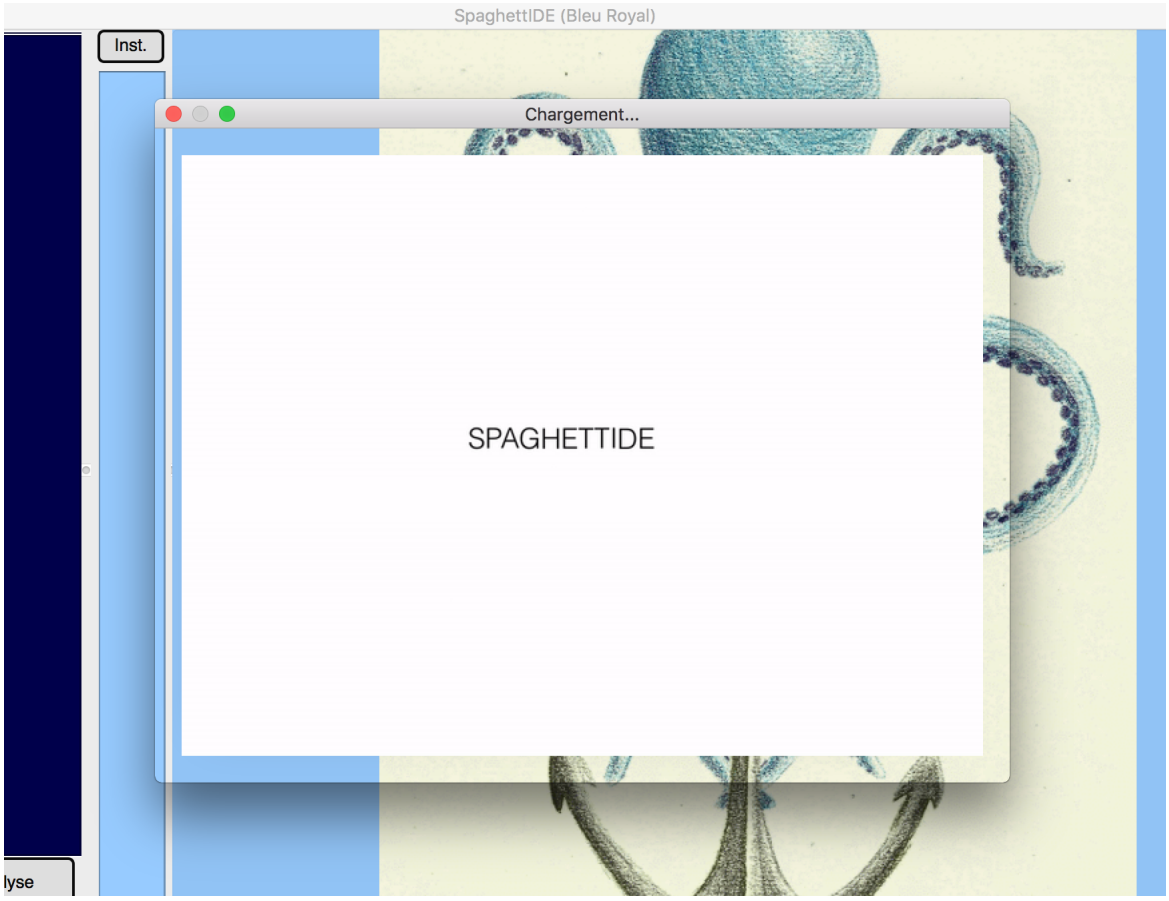

FIGURE 8 – Écran de chargement au démarrage

4.4 La numérotation des lignes

La numérotation des lignes est également sélectionnable via le menu et elle reste tout le temps active après son activation et se complète progressivement après l'ouverture et l'écriture d'un fichier.

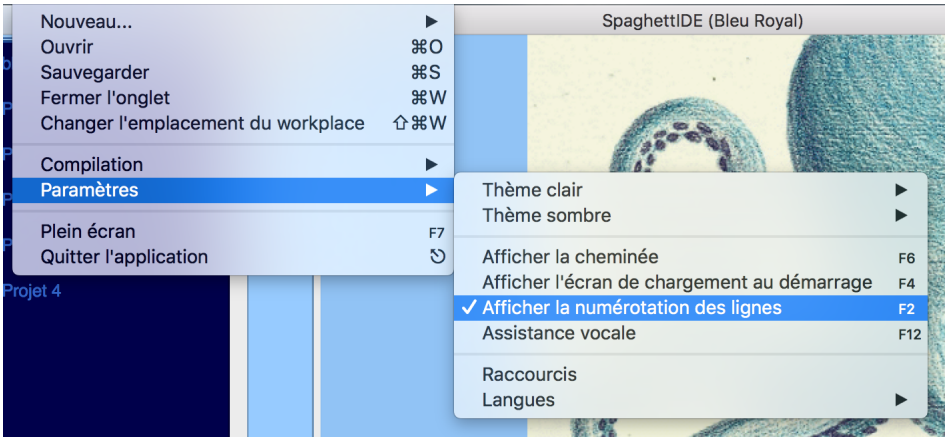

FIGURE 9 – Interface graphique de la sélection de la numérotation des lignes

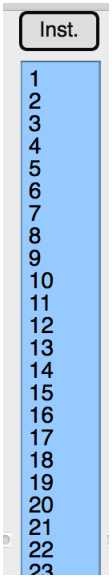

FIGURE 10 – Numérotation des lignes

5 Exemples concrets de l'utilisation du xml dans notre IDE

Nous allons voir dans cette partie des exemples concrets de l'utilisation des fonctionnalités dans l'IDE. Nous pouvons voir dans l'image ci-dessous la fonction permettant l'activation ou la désactivation de l'assistance vocale. Nous pouvons constater l'utilisation du dictionnaire stockant la valeur de la balise "assistance_vocale" afin d'écrire la valeur voulue pour son utilisation.

```
def assist_voc(self):
    """
    Active ou désactive l'assistance vocale et écrit la configuration actuelle dans un fichier XML.
    """
    if "darwin" in sys.platform:
        configuration = open_xml("conf.xml")
        if configuration['assistance_vocale'] == 'True':
            self.status_message(get_text("status_voc_des"))
            write_xml("conf.xml", "assistance_vocale", "False")
        else:
            write_xml("conf.xml", "assistance_vocale", "True")
            self.status_message(get_text("status_voc_activ"))
```

FIGURE 11 – Exemple de l'assistance vocale

Nous pouvons voir dans l'image ci-dessous la fonction permettant l'activation ou la désactivation de la numérotation des lignes. Nous pouvons constater encore une fois l'utilisation du dictionnaire stockant la valeur de la balise "numerote_lines" afin d'écrire la valeur voulue pour son utilisation.

6 Les projets et le xml

Nous allons voir dans cette partie l'utilisation du xml avec les projets. Encore une fois, nous utilisons le dictionnaire stockant les différentes balises et leurs valeurs grâce à la fonction qui parcours un fichier xml et stocke ses données "open_xml". Dans l'image ci-dessous, nous pouvons voir par exemple l'utilisation de project["name"] pour récupérer le nom du projet, de même avec le langage, la localisation, la date de création et le nombre de fichiers.

```python
def show_line_column(self):
    """
    Affiche et masque la colonne de numérotation des lignes.
    """
    if configuration['numerote_lines'] == 'False':
        write_xml("conf.xml", "numerote_lines", "True")
    else:
        write_xml("conf.xml", "numerote_lines", "False")

    self.is_show_line = not self.is_show_line
    if self.is_show_line:
        self.line_tab.setMaximumWidth(60)
    else:
        self.line_tab.setMaximumWidth(1)
    self.splitter.update()
```

FIGURE 12 – Exemple de la numérotation des lignes

```python
project_name = chemin.split("/")[-1]
path = "%s/%s.xml"%(chemin, project_name)
update_nb_files(parent, path, project_name)
project = open_xml(path)
self.name = QLabel(get_text("project_name") + project["name"])
self.language = QLabel(get_text("project_language") + project["language"])
self.location = QLabel(get_text("project_location") + chemin)
self.creation_date = QLabel(get_text("project_creation_date") + project["creation_date"])
self.number_files = QLabel(get_text("project_number_files") + project["number_files"])
```

FIGURE 13 – L'utilisation du xml dans les projets

Voici l'aperçu d'un fichier xml d'un projet avec les différentes valeurs associées aux balises stockées dans ce dernier :

```xml
<project>
    <name>Projet</name>
    <creation_date>2017-04-10 23:27:48.479969</creation_date>
    <language>C</language>
    <location>/Users/alexiscarreau/workplace/Projet</location>
    <number_files>5</number_files>
    <compil> </compil>
    <compil_json> </compil_json>
</project>
```

FIGURE 14 – Exemple de fichier xml de projet

7 Conclusion

En conclusion, nous pouvons dire que le XML est vraiment très pratique et nous l'avons utilisés dans la mesure du possible dans notre IDE pour permettre une facilité d'exécution des différentes fonctionnalités le composant. En amélioration, on pourrait imaginer la prise en compte total de tous les projets dans le fichier xml déjà alloué avec sa mise à jour lors de la suppression ou la modification des informations de projet.