Apprendre Python – Partie 3 – Automatiser Microsoft Office

2012-04-17 by Thomas

http://bores.fr/blog/2012/04/utiliser-python-pour-automatiser-microsoft-office/

En ce moment, rédaction de spécifications oblige, je fais une overdose de Microsoft Office au boulot. Pour me simplifier la vie, j'ai essayé d'apprendre un maximum de raccourcis pour Word et Excel. Mais comme je dois travailler sur deux PC différents dont l'un en français et le second en anglais et que chez Microsoft on n'a rien trouvé de mieux que de changer les raccourcis en fonction de la langue du système d'exploitation, je fais souvent de mauvaises manipulations. D'autre part, qui dit spécifications, dit exigences (celui qui vous dit l'inverse n'a jamais codé de sa vie) et donc Requirement Management. Sous ce terme à faire frémir toute personne qui a un jour écrit des spécifications, se cache la gestion et le suivi des exigences. Une tâche rébarbative, qui l'est encore plus quand on gère les exigences avec des outils de bureautique pas conçus pour, au lieu d'outils dédiés comme celui-ci ou celui-là.

Résultat, je perds du temps précieux sur des tâches de mise en forme, de copier-coller et de relecture, temps précieux, qui serait bien mieux utilisé, en cette période d'élections, à refaire le monde avec les collègues lors de pauses café allongées, pendant que mon PC s'occupe des tâches rébarbatives à ma place!

C'est pourquoi, j'ai cherché à automatiser un maximum de choses. Alors, bien sûr la première idée qui vient à l'esprit lorsque l'on parle de la suite MS Office, c'est le VBA (Visual Basic for Application). Solution inenvisageable pour mon cas:

- Technologie Microsoft, en général, je ne suis pas fan, car ça ne fonctionne que dans des cadres très précis et avec des technologies qui ne sont pas standard. De plus, venant du monde Unix, je suis souvent perdu avec le paradigme des technologies made in Microsoft (je déteste Visual Studio).
- Les applications VBA sont, en général, un ensemble de macros embarquées dans un document Office. Besoin d'avoir une solution générique.
- Le VBA est en voie de disparition et est amené à être remplacé par Visual Studio Tools for Application (VSTA).

Python, étant le langage de script que j'utilise le plus souvent, j'ai fait des recherches dans ce sens et j'ai trouvé de quoi répondre à ma problématique: pywin32. Cette librairie, qui nécessite au préalable l'installation de python, est librement téléchargeable ici.
Pywin32 offre la possibilité de s'interfacer avec l'API Win32 et la couche COM (Component Object Layer) de Windows. Pour rappel, COM est une interface IPC (Inter Process Communication) qui permet le dialogue entre les différentes applications Windows qui l'implémentent.

Il en résulte que Pywin32 ne permet de s'abstraire totalement des technologies Microsoft, mais plutôt d'offrir le moyen de s'y connecter facilement et de gagner du temps sur le développement en partant d'un langage connu, python, qui offre une grammaire souple et des fonctionnalités de base avancées. Encore mieux, Python, via le nombre de ses librairies disponibles sur l'internet, permettra au programmateur d'enrichir rapidement ses scripts et même de créer de véritables petits programmes. En ajoutant par exemple : une interface graphique (pyqt), la création de diagrammes (pyX) , une connexion à une base de données (SqlAlchemy)…

Commençons les choses sérieuses. Premièrement, vous aurez besoin des documentions de Microsoft:

- Word 2007 Developer Reference
- Word Object Model Reference
- Excel 2007 Developer Reference
- Excel Object Model Reference

Puis de la documentation de la librairie pywin32:

- Pywin32 online documentation copy, cependant n'hésitez à regarder la documentation (fichier .chm) installée avec pywin32 surement plus à jour.
- Win32 How do I. Trucs et astuces en python pour interagir avec Windows.

Maintenant passons au code. Il est possible de démarrer facilement Word de cette manière:

```
word = win32.gencache.EnsureDispatch('Word.Application');
```

de même pour Excel:

```
excel = win32.gencache.EnsureDispatch('Excel.Application')
```

A partir de là, il est aussi très simple de créer ou d'ouvrir un document, de faire les travaux dessus, de le fermer et enfin de fermer l'application.
Exemple avec word, ouverture de document:

```
# Start word and open word document
word = win32.gencache.EnsureDispatch('Word.Application');
doc = word.Documents.Open("C:\myWordFile.doc")
word.Visible = False
sleep(1)

# Do the work

# Close document and quit Microsoft Word
doc.Close(0, 1) # 0 = do not saved changes, 1 = original format
word.Application.Quit()
```

Exemple avec excel, création de document:

```
# Start excel and create a new document with a sheet named mySheet
excel = win32.gencache.EnsureDispatch('Excel.Application')
xls = excel.Workbooks.Add()
ws = xls.ActiveSheet
ws.Name = "mySheet"
```

```
excel.Visible = False
sleep(1)

# Do the work

# Save file, close it and quit excel
xls.SaveAs(excelFilePath)
xls.Close()
excel.Application.Quit()
```

Finissons avec un exemple plus concret.

Expression du besoin: écrire une fonction qui analyse un document word et en extrait les numéros d'exigences. Dans le document word, les exigences sont contenues dans la première colonne du tableau et respecte l'expression régulière contenue dans la variable reqRegExPattern. Le chemin du fichier est fournie via le paramètre d'entrée filePath

Il faut aussi savoir, je l'ai remarqué en développant, que chaque cellule d'un tableau word se termine par le caractère

```
WORD_END_CELL_SEPARATOR = u'\x07'
```

Voici la solution que je propose:

```
def wordExtractRequirements(filePath):
    reqList = []
# Start Microsoft Word, open the file and hide word
    word = win32.gencache.EnsureDispatch('Word.Application');
    doc = word.Documents.Open(filePath)
    word.Visible = False
    sleep(1)

# Parse all tables in word document and look if it contains a requirement
# Save the founded requirement in a table structure in memory
    for table in doc.Tables:
       for i in range(1, table.Rows.Count+1):
            cellContent = unicode(table.Cell(i,
1)).rstrip(WORD_END_CELL_SEPARATOR).rstrip().lstrip()
            if re.match(reqRegExPattern, cellContent):
                reqList.append(reqID)

# Close document and quit Microsoft Word
    doc.Close(0, 1) # 0 = do not saved changes, 1 = original format
    word.Application.Quit()

    return reqList
```

Pour finir, on peut imaginer écrire une fonction qui sauvegarderait ces numéros d'exigence dans un fichier excel, le paramètre d'entrée reqList contient la liste des exigences à sauvegarder:

```
def excelExportRequirements(reqList):
# Start Microsoft Excel, create a new file
    excel = win32.gencache.EnsureDispatch('Excel.Application')

# Add a new sheet and save requirements in it
    xls = excel.Workbooks.Add()
    ws = xls.ActiveSheet
    ws.Name = "Requirements List"
```

```
    excel.Visible = False
    sleep(1)

    i = 1;
    for req in reqList:
        ws.Cells(i, 1).Value = req[0]
        i += 1

# Save file, close it and quit excel
    xls.SaveAs(excelFilePath)
    xls.Close()
    excel.Application.Quit()
```

That’s all folks!

Article librement inspiré de ce post.