Python tames Excel



guy.dalberto@alcos.fr
June 2005

Alcos

I am here to show you how Python can be an excellent driver for Excel

Context

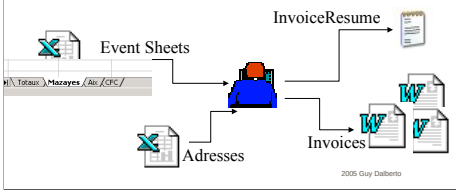
- Speaker :
  - Guy Dalberto, French, born 1951, married, two grown sons,
  - living and working in Ardèche, near Valence 100 km south of Lyon
  - INPG ingénieur 1974 in fundamental physics
  - writing software since 1975 : real-time, Telecom, Test&Measure
  - preferred languages : Python, C++(STL, Boost)
  - 1992-2005 freelance (sole owner of Alcos)
  - hobbies : walks, mountain-ski, cycling, orienteering
- Valence Sports Orientation (VSO)
  - the orienteering club of Valence, Drôme
  - invoices of event with expenses are done with Excel
  - There was a lot of repetitive work



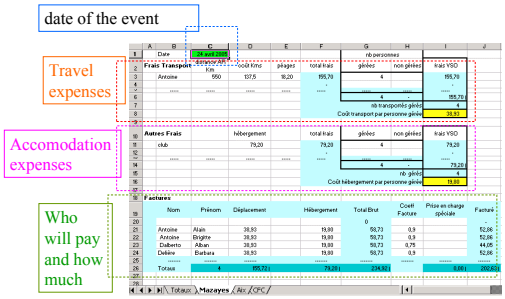
+ I do a job I like in a very fine place
+ I don't lose time in daily travels
+ direct TGV train to Lyon, Paris and Marseille
+ I hate repetitive work

VSO invoice repetitive Data flow

- Input File :
  - 1 Excel file : containg from 1 to N Event description sheets
  - 1 Excel File : the adresses and emails of VSO members
- Output :
  - 1 Text File : resume of all invoices
  - N Word files : invoices



input : Mazayes VsoEvent Sheet

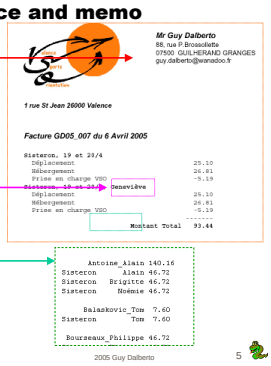


Each Vso Event Sheet contains info about :

+ the date of the event
+ the travel expenses
+ the accomodation expenses (hotel, restaurant, ..)
+ which members (or non members) were there and what should be invoiced to each of them

outputs : VSO invoice and memo

- Person that must pay the Invoice
- Info about Events date, place, invoice items
- Items can concern another person
- The memo text file contains a resume of the invoices

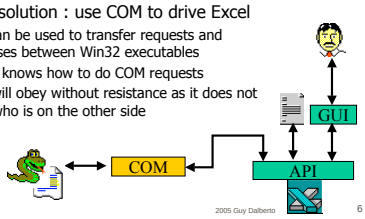


One invoice can be relative to :

+ multiple events
+ different persons (some of whom are not mebers)
+ with different club participation

Exploring the solutions

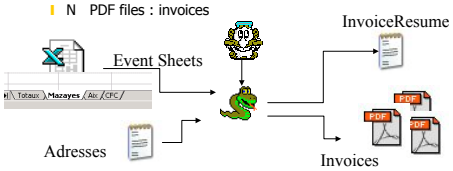
- Poor solution : a Visual Basic script
  - the Excel engine is most often used through the GUI interface
  - has a powerful API through which you can request all its services from a VB script
- Powerful solution : use COM to drive Excel
  - COM can be used to transfer requests and responses between Win32 executables
  - Python knows how to do COM requests
  - Excel will obey without resistance as it does not know who is on the other side



+ two other solutions, do nothing, write en entire Python application
+ but you either : don't know how to write Visual Basic scripts, would prefer to forget what you knew

VSO invoice automated Data flow

- Input File :
  - 1 Excel file : N Event description Worksheets
  - 1 Text File : the adresses and emails of VSO members, created by exporting manually an Excel file in CSV format
- Output :
  - 1 Text File : resume of all invoices
  - N PDF files : invoices



COM = M$ Component Object Model

- COM is a WIN32 technology :
  - that is used to mediate between Server Objects and Client Applications
  - server objects implement Interfaces that are used by the client
  - an application can offer well-known interfaces : Copy/Cut/Paste, Drag/Drop
  - an application can define and implement new Interfaces
  - many applications offer a GUI access and a COM-conforming API
- CORBA is the best known alternative
- COM works tolerably well, at least on the same computer

Microsoft has promoted different technologies : DDE, OLE, VBX, COM, ActiveX, DCOM, COM+, .NET

COM is very difficult to program, but Mark Hammond did a lot for you

Quick & Dirty : write helloworld.xls

```
import win32com.client

def writeHelloWorldExcelFile(filePath):
    xlApp = win32com.client.Dispatch( "Excel.Application" )
    xlBook = xlApp.Workbooks.Add( )
    xlSheet = xlBook.Worksheets(1)
    xlSheet.Name = "Adresses"
    xlSheet.Cells(1,1).Value = "Hello World!"
    xlBook.SaveAs(filePath)
    xlBook.Close(SaveChanges=0)
```

- will :
  - create an .xls file
  - rename the first sheet
  - set value of first cell

Python for Win32 Extensions

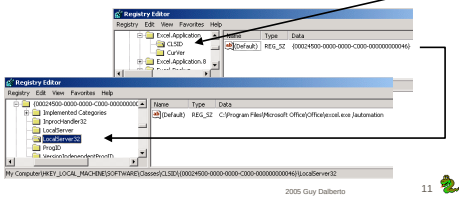
- Special package, see Reference slide
- What you get :
  - Direct access to the Win32 Native API
  - PythonWin : a Python IDE application (with a debugger)
  - **generic COM access**
- It means that you can write in Python
  - COM clients :
    import win32com.client
  - COM servers
    import win32com.server

create a COM object through which we can access an Excel Application
create an Excel WorkBook, and keep COM instance to access it
create a COM object through which we can access the first sheet
change name of that WorkSheet
change content of Cell(1,1)
save the Workbook in a file
close the Workbook (and the file we created)

Python requesting an Excel COM server

xlApp = win32com.client.Dispatch("Excel.Application")

- a COM server registers itself in the Windows Registry
- the ProgID of the server is Excel.Application
- its CLSID is {00024500-0000-0000-C000-000000000046}
- if there is no active Server, Windows starts the executable defined by LocalServer32, i.e. excel.exe

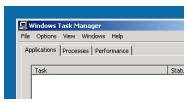


COM uses globally unique identifiers—128-bit integers that are guaranteed to be unique in the world across space and time—to identify every interface and every component object class. These globally unique identifiers are UUIDs (universally unique IDs) as defined by the Open Software Foundation's Distributed Computing Environment.

A GUID is generated by a complex algorithm, that uses the current Date/Time and the Network interface ID

Excel.exe is a bit shy, it starts hidden

- There is nothing
  - in the Task Bar
  - in the Application Tab of the Task Manager



You MUST ALLWAYS close the Workbook

```
def writeHelloWorldExcelFile(filePath):
    .....
    xlBook.Close(SaveChanges=0)
```

- If you don't close the Workbook :
  - the hidden Excel process is still running
  - it will keep the file open
  - you will have to kill it without mercy from the Processes tab of the Task Manager

+ win32com.client offers you all the power and drawbacks of the API of the Server

+ Excel Server presupposes that you will close the Workbook

+ del xlBook # does not Close the Workbook

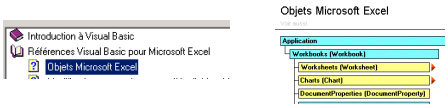
+ You must protect yourself against Exceptions

+ The previous code should be written with a try/finally

or a Wrapper class with a __del__ method

Understanding Excel Object model

- Once you have an Excel Application instance, you can read/write its properties and call its methods
- Through win32com.client you access them easily, under the same names and with the same parameters
- Excel must be installed with Visual Basic for Application Help Reference
- An Application has a Workbooks property that is a Collection of Workbook



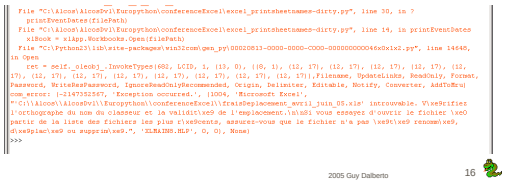
Quick & Dirty : Print Date of all events

```
def printEventDates(filePath):
    xlApp = win32com.client.Dispatch('Excel.Application')
    # create an Excel WorkBook that loads its data from file
    xlBook = xlApp.Workbooks.Open(filePath)
    # print Date of all events
    nbSheets = xlBook.Worksheets.Count
    for ish in range(1, nbSheets+1): # VB Collection from 1 to N
        xlSheet = xlBook.Worksheets(ish)
        eventDate = xlSheet.Cells(1,3).Value
        print "%d : %s %s" % (ish, xlSheet.Name, eventDate)
    xlBook.Close(SaveChanges=0) # don't forget to close
```


That code is really dirty, why ?

Some of the problems waiting to happen

- We are not protected against Exceptions
- We are going to copy the same code until the end of times
- We access the content of the Sheet by magic row and column numbers : Cells(1,3)
- We are using 1 based indexes
- When we try to open an file that does not exists, we get an unreadable traceback



row, column, sheetIndex

XslBook : an Excel Client wrapper

- We need safe and simple access to a Workbook
- The Right Way (TM) to do that is to :
  - write a little helper class : XslBook, in vso/excel.py
  - Unit Test it
  - (re)raise readable Exceptions
- We will be able to write code like :

```
wkb = XslBook(filePath)
for name in wkb.getSheetNames( ):
    ...
```

- Or if we want to create a new file :

```
wkb = XslBook( )
....
wkb.save(filePath)
```


Initializing an XslBook

```
def __init__ (self, filePath=None) :
    """Load Workbook from file or create an empty one."""
    xlApp = win32com.client.Dispatch('Excel.Application')
    self.filePath = filePath # path of current file, can be None
    # load Workbook from file or create an empty one
    if filePath:
        self.book = xlApp.Workbooks.Open(filePath)
    else:
        self.book = xlApp.Workbooks.Add( )
    # current sheet is first one
    self.sheet = self.book.Worksheets(1)
```


+ Instance does not keeps reference of Excel Application
+ each instance of XslBook will manage one Workbook

Closing an XslBook

```
def __del__ (self):
    """call close with protection against exceptions"""
    try:
        self.close()
    except Exception, e:
        print "XslBook.__del__, exception" , e

def close (self):
    """Close Book without saving it in file."""
    self.sheet = None
    if not self.book is None:
        self.book.Close(SaveChanges=0)
        self.book = None
```


Unit Test : the fastest way to success

- We begin to exercise our XslBook class with code in vso/test_excel.py
  - normal and exceptional cases
  - What happens if the file does not exists ?

```
class XslBookTestCase(unittest.TestCase):

    def setUp(self):
    def testOpenMissingFile(self):
    def testSaveWithoutFilePath(self):
    def testSaveInvalidFolder(self):
    def testCreateExcelFile(self):
    def test_getSheetNames(self):
    def test_selectSheet(self):
    def test_getNamedCell(self):
    def test_getMissingNamedCell(self):

def testOpenMissingFile(self):
    self.failUnlessRaises(AssertionError, XslBook, "missingfile.xls")
```

COM unreadable Exceptions

- Each time we get an ureadable COM exception, we try to insert an assert with a readable message :

```
    if filePath:
        assert os.path.isfile(filePath), (
                "File not found: <<%s>>" % filePath )
        self.filePath = filePath
        self.book = xlApp.Workbooks.Open(filePath)
```

```
File "C:\Alcos\AlcosDv1\Europython\conferenceExcel\printsheetnames.py", line 23, in ?
  printEventDates(filePath)
File "C:\Alcos\AlcosDv1\Europython\conferenceExcel\printsheetnames.py", line 14, in print
  wkb = XslBook(filePath)
File "C:\Alcos\AlcosDv1\Europython\conferenceExcel\vso\excel.py", line 58, in __init__
  assert os.path.isfile(filename), "File not found: <<%s>>" % filename
AssertionError: File not found:
<<C:\Alcos\AlcosDv1\Europython\conferenceExcel\vso\qsfraisDeplacement_avril_juin_05.xls>>
```


XslBook is sprinkled with asserts

```
def save(self, filePath=None):
    """Save WorkBook as an Excel file."""
    if filePath:
        self.filePath = filePath
    assert self.filePath, "you must specify a filePath"
    dirPath = os.path.dirname(self.filePath)
    assert os.path.isdir(dirPath), ( "%r is not a directory" %
            dirPath )
    self.book.SaveAs(self.filePath)
```


XslBook : finding the Data

- Accessing our Data with magic coordinates is really unproper
  `eventDate = xbw.sheet.Cells(1,3).Value`
  - we could use variables in the Python code, but each time somebody inserts a column or row our code will crash
- An Excel Range :
  - specify a rectangular array of Cells from 1x1 to MxN
  - can be given a name
  - we can use name to read or write value(s)
    `eventDate = xbw.sheet.Range( 'date' ).Value`
- An Excel name scope can be global or local to a sheet
  - it can be defined on many sheets
  - it is a kind of attribute name
  - if it is used without prefix it concerns the current sheet
  - it can be used with a sheet name prefix Mazayes!date

XslBook : range usage

- if a name does not exists, trying to find a range will raise an unreadable COM exception, we can catch it and raise another one

```
def getNamedRange(self, rangeName):
    try:
        return self.sheet.Range(rangeName)
    except Exception, e:
        raise KeyError, ( "range <<%s>> not found" % rangeName)
```

- getting the position of a cell

```
def getNamedRangePosition(self, rangeName):
    """Return tuple (ixRow, ixCol) of top left cell in Range."""
    range = self.getNamedRange(rangeName)
    return (range.Row - 1, range.Column - 1)
```

Data Format, reading raw values

- COM specifies a list of types that are used for the transfer from Server to Client
- win32com gives us the data in types that are compatible with that format

```
def getCellValueRaw(self, rowIndex, columnIndex):
    """Return the value of one cell in COM format.
       rowIndex, columnIndex -- values from 0 to N - 1
    """
    return self.sheet.Cells(rowIndex +1,
                columnIndex + 1).Value
```


+ setting a value is as natural

Data Format, reading converted values

- for our application it will be easier if we read the data in our usual types

```
def getCellValue(self, rowIndex, columnIndex):
    return convertXslCellValueToPy(
        self.getCellValueRaw(rowIndex, columnIndex))

def getNamedCellValue(self, cellName):
    return convertXslCellValueToPy(
        self.getNamedRange(cellName).Value)
```

- convertXslCellValueToPy will convert
  Unicode strings -> Latin-1
  TimeType -> int (seconds)
  monetary tuple -> float
- M$ monetary type is returned as tuple (0, 10000 * floatval)

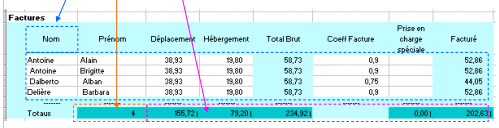
Much better : Print Date of VSO events

```
def printEventDates(filePath):
    """Print the event dates of the Worksheets of that file."""
    wkb = XslBook(filePath)
    for sheetName in wkb.getSheetNames( ):
        if sheetName == 'Totaux' :
            continue
        wkb.selectSheet(sheetName)
        eventDate = wkb.getNamedCellValue( 'Date' )
        print "%s %s" % (sheetName, eventDate)
```


Our code is no longer Quick & Dirty

VSOEventSheet : Invoice Items

- We use some local names to read the invoice items :
  - nbParticipants gives us the Number of **InvoiceItems**
  - listeFactures position is used to locate the array of items
  - other named cells gives us sub-totals that we will use to check



VSO, Generating the Invoices

- You request work from the Generator

```
class Generator:
    def __init__ (self, fileSuffixName, comptaFolder, invoiceFolder):
    def initAdresses (self, csvFileName):
    def readExcelFile (self):
    def createInvoices (self,factPrefix,numFirstInvoice,dateInvoices):
    def saveInvoicesInPdfFiles (self, imageFolder):
    def _cleanInvoiceFolder (self):
    def _createResumedInfos (self):
    def _printDetail (self):
    def _printTotaux (self):
```


VSO, other classes

- Helper classes

```
class EventFileReader:
    def readVsoEvents (self, excelFilePath):  # Return a list of VsoEvent
class PdfWriter:
    def __init__ (self, imageFolder):
    def save (self, invoice, adresse, pdfFilePath):
    def drawLogo (self):
    def drawSpecifics (self, invoice, adresse):
```

- and data enveloppe classes

```
class InvoiceItem:
    def __init__ (self, date, lieu, nom ,prenom, dictVals):
    def  key (self): return "%s_%s" % (self.nom, self.prenom)
class VsoEvent (dict):  # dictionnary of InvoiceItem
class Invoice(list): # List of InvoiceItem
class Adresse: # Holds address and email info for somebody
```


+ many infos ave an acees by key nome_prenom

References :

- Python Win32 Extensions
  - http://sourceforge.net/projects/pywin32/
    Win32 installer : pywin32-204.win32-py2.4.exe or later
  - Mark Hammond book : Python Programming on Win32
    http://www.oreilly.com/catalog/pythonwin32/
- Excel
  - Excel help Menu/Contents/Visual basic Reference for M$ Excel
  - Louise Pryor : see Names in Excel
    http://www.louisepryor.com/showThemes.do
  - Use Global and Local References in Formulas in Excel 2000
    http://support.microsoft.com/?kbid=274504
- PDF library : http://www.reportlab.org/downloads.html
  - download, and install ReportLab_1_20.zip or later
- VSO invoice automation : samples on the EuroPython site

You can download the VSO invoice system samples