Quick Start to Client side COM and Python

file:///D:/myTools/Python/Python36/Lib/site-packages/win32com/HTML/QuickStartClientCom.html

Introduction

This documents how to quickly start using COM from Python. It is not a thorough discussion of the COM system, or of the concepts introduced by COM.

Other good information on COM can be found in various conference tutorials - please see the collection of Mark's conference tutorials

For information on implementing COM objects using Python, please see a Quick Start to Server side COM and Python

In this document we discuss the following topics:

Quick Start

To use a COM object from Python

```
import win32com.client
o = win32com.client.Dispatch("Object.Name")
o.Method()
o.property = "New Value"
print o.property
```

Example

```
o = win32com.client.Dispatch("Excel.Application")
o.Visible = 1
o.Workbooks.Add() # for office 97 - 95 a bit different!
o.Cells(1,1).Value = "Hello"
```

And we will see the word "Hello" appear in the top cell.

How do I know which methods and properties are available?

Good question. This is hard! You need to use the documentation with the products, or possibly a COM browser. Note however that COM browsers typically rely on these objects registering themselves in certain ways, and many objects to not do this. You are just expected to know.

The Python COM browser

PythonCOM comes with a basic COM browser that may show you the information you need. Note that this package requires Pythonwin (ie, the MFC GUI environment) to be installed for this to work.

There are far better COM browsers available - I tend to use the one that comes with MSVC, or this one!

To run the browser, simply select it from the Pythonwin *Tools* menu, or double-click on the file *win32com\client\combrowse.py*

Static Dispatch (or Type Safe) objects

In the above examples, if we printed the '`repr(o)`' object above, it would have resulted in

```
<COMObject Excel.Application>
```

This reflects that the object is a generic COM object that Python has no special knowledge of (other than the name you used to create it!). This is known as a "dynamic dispatch" object, as all knowledge is built dynamically. The win32com package also has the concept of *static dispatch* objects, which gives Python up-front knowledge about the objects that it is working with (including arguments, argument types, etc)

In a nutshell, Static Dispatch involves the generation of a .py file that contains support for the specific object. For more overview information, please see the documentation references above.

The generation and management of the .py files is somewhat automatic, and involves one of 2 steps:

- Using *makepy.py* to select a COM library. This process is very similar to Visual Basic, where you select from a list of all objects installed on your system, and once selected the objects are magically useable.

or

- Use explicit code to check for, and possibly generate, support at run-time. This is very powerful, as it allows the developer to avoid ensuring the user has selected the appropriate type library. This option is extremely powerful for OCX users, as it allows Python code to sub-class an OCX control, but the actual sub-class can be generated at run-time. Use *makepy.py* with a -i option to see how to include this support in your Python code.

The *win32com.client.gencache* module manages these generated files. This module has some documentation of its own, but you probably don't need to know the gory details!

How do I get at the generated module?

You will notice that the generated file name is long and cryptic - obviously not designed for humans to work with! So how do you get at the module object for the generated code?

Hopefully, the answer is *you shouldn't need to*. All generated file support is generally available directly via *win32com.client.Dispatch* and *win32com.client.constants*. But should you ever really need the Python module object, the win32com.client.gencache module has functions specifically for this. The functions GetModuleForCLSID and GetModuleForProgID both return Python module objects that you can use in your code. See the docstrings in the gencache code for more details.

To generate Python Sources supporting a COM object

Example using Microsoft Office 97.

Either:

- Run '`win32com\client\makepy.py`' (eg, run it from the command window, or double-click on it) and a list will be presented. Select the Type Library '`Microsoft Word 8.0 Object Library`'
- From a command prompt, run the command '`makepy.py "Microsoft Word 8.0 Object Library"`' (include the double quotes). This simply avoids the selection process.
- If you desire, you can also use explicit code to generate it just before you need to use it at runtime. Run '`makepy.py -i "Microsoft Word 8.0 Object Library"`' (include the double quotes) to see how to do this.

And that is it! Nothing more needed. No special import statements needed! Now, you simply need say

```
>>> import win32com.client
>>> w=win32com.client.Dispatch("Word.Application")
>>> w.Visible=1
>>> w
<win32com.gen_py.Microsoft Word 8.0 Object Library._Application>
```

Note that now Python knows the explicit type of the object.

Using COM Constants

Makepy automatically installs all generated constants from a type library in an object called *win32com.clients.constants*. You do not need to do anything special to make these constants work, other than create the object itself (ie, in the example above, the constants relating to *Word* would automatically be available after the `w=win32com.client.Dispatch("Word.Application"`) statement.

For example, immediately after executing the code above, you could execute the following:

```
>>> w.WindowState = win32com.client.constants.wdWindowStateMinimize
```

and Word will Minimize.