## Introduction

win32com attempts to provide a seamless COM interface and hide many COM implementation details, including the use of COM VARIANT structures. This means that in most cases, you just call a COM object using normal Python objects as parameters and get back normal Python objects as results.

However, in some cases this doesn't work very well, particularly when using "dynamic" (aka late-bound) objects, or when using "makepy" (aka early-bound) objects which only declare a parameter is a VARIANT.

The win32com.client.VARIANT object is designed to overcome these problems.

## Drawbacks

The primary issue with this approach is that the programmer must learn more about COM VARIANTs than otherwise - they need to know concepts such as variants being *byref*, holding arrays, or that some may hold 32bit unsigned integers while others hold 64bit signed ints, and they need to understand this in the context of a single method call. In short, this is a relatively advanced feature. The good news though is that use of these objects should never cause your program to hard-crash - the worst you should expect are Python or COM exceptions being thrown.

## The VARIANT object

The VARIANT object lives in win32com.client. The constructor takes 2 parameters - the 'variant type' and the value. The 'variant type' is an integer and can be one or more of the pythoncom.VT\_\* values, possibly or'd together.

For example, to create a VARIANT object which defines a byref array of 32bit integers, you could use:

>>> from win32com.client import VARIANT  
>>> import pythoncom  
>>> v = VARIANT(pythoncom.VT\_BYREF | pythoncom.VT\_ARRAY | pythoncom.VT\_I4,  
... [1,2,3,4])  
>>> v  
win32com.client.VARIANT(24579, [1, 2, 3, 4])  
>>>

This variable can then be used whereever a COM VARIANT is expected.

## Example usage with dynamic objects.

For this example we will use the COM object used for win32com testing, PyCOMTest.PyCOMTest. This object defines a method which is defined in IDL as:

HRESULT DoubleInOutString([in,out] BSTR \*str);

As you can see, it takes a single string parameter which is also used as an "out" parameter - the single parameter will be updated after the call. The implementation of the method simply "doubles" the string.

If the object has a type-library, this method works fine with makepy generated support. For example:

>>> from win32com.client.gencache import EnsureDispatch  
>>> ob = EnsureDispatch("PyCOMTest.PyCOMTest")  
>>> ob.DoubleInOutString("Hello")  
u'HelloHello'  
>>>

However, if makepy support is not available the method does not work as expected. For the next example we will use DumbDispatch to simulate the object not having a type-library.

>>> import win32com.client.dynamic  
>>> ob = win32com.client.dynamic.DumbDispatch("PyCOMTest.PyCOMTest")  
>>> ob.DoubleInOutString("Hello")  
>>>

As you can see, no result came back from the function. This is because win32com has no type information available to use, so doesn't know the parameter should be passed as a byref parameter. To work around this, we can use the VARIANT object.

The following example explicitly creates a VARIANT object with a variant type of a byref string and a value 'Hello'. After making the call with this VARIANT the value is updated.

>>> import win32com.client.dynamic  
>>> from win32com.client import VARIANT  
>>> import pythoncom  
>>> ob = win32com.client.dynamic.DumbDispatch("PyCOMTest.PyCOMTest")  
>>> variant = VARIANT(pythoncom.VT\_BYREF | pythoncom.VT\_BSTR, "Hello")  
>>> variant.value # check the value before the call.  
'Hello'  
>>> ob.DoubleInOutString(variant)  
>>> variant.value  
u'HelloHello'  
>>>

## Usage with generated objects

In most cases, objects with makepy support (ie, 'generated' objects) don't need to use the VARIANT object - the type information means win32com can guess the right thing to pass. However, in some cases the VARIANT object can still be useful. Imagine a poorly specified object with IDL like:

HRESULT DoSomething([in] VARIANT value);

But also imagine that the object has a limitation that if the parameter is an integer, it must be a 32bit unsigned value - any other integer representation will fail.

If you just pass a regular Python integer to this function, it will generally be passed as a 32bit signed integer and given the limitation above, will fail. The VARIANT object allows you to work around the limitation - just create a variant object VARIANT(pythoncom.VT\_UI4, int\_value) and pass that - the function will then be called with the explicit type you specified and will succeed.

Note that you can not use a VARIANT object to override the types described in a type library. If a makepy generated class specifies that a VT\_UI2 is expected, attempting to pass a VARIANT object will fail. In this case you would need to hack around the problem. For example, imagine ob was a COM object which a method called foo and you wanted to override the type declaration for foo by passing a VARIANT. You could do something like:

>>> import win32com.client.dynamic  
>>> from win32com.client import VARIANT  
>>> import pythoncom  
>>> dumbob = win32com.client.dynamic.DumbDispatch(ob)  
>>> variant = VARIANT(pythoncom.VT\_BYREF | pythoncom.VT\_BSTR, "Hello")  
>>> dumbob.foo(variant)

The code above converts the makepy supported ob into a 'dumb' (ie, non-makepy supported) version of the object, which will then allow you to use VARIANT objects for the problematic methods.