# Using COM Without Registration

imagiro, 10 Oct 2015 CPOL 2.5K 97 11

A small class for using COM DLL modules without registering them

**Download sample code - 24.1 KB**

**Download ComLibrary - 1.9 KB**

## Introduction

This tip describes how to use a COM DLL without registering it. You will find a small class here which you can use in your projects.

C++ developers who have heard about COM but don't know it, might think "Nuh, all the stuff I have to implement - too complicated". But when it comes to modularization, COM provides an excellent mechanism to deal with components imported from DLLs.

If you are one of these developers, you should probably read Alex Blekhman's article about how to export classes from a DLL. It leads nicely to the concept of COM - exporting interfaces.

When you reached the point where you think that using interfaces to export classes is really a good idea, you might notice that COM actually provides you with all the mechanisms you need to manage the lifetime of your objects and your modules and to create instances. You might also notice that by exporting a few functions from your DLL, you can create a COM DLL that you can easily use from your application.

One downside is there though: Normally a COM DLL has to be registered so the OS can load it when you say "create an instance of this or that class". You might not want that, maybe because you don't want your application to require any installation, maybe you don't want that your DLL can be used (easily) by other applications. So that's what this tip is about: Using components from your DLLs without registering.

## Using the Code

Usually, when you want to instantiate a COM object, you would use:

Hide Copy Code

```
HRESULT CoCreateInstance(
  REFCLSID  rclsid,
  LPUNKNOWN pUnkOuter,
  DWORD     dwClsContext,
  REFIID    riid,
  LPVOID    *ppv
);
```

COM will lookup the `CLSID` in the registry, see which module provides this class, load the module (DLL) and ask the module to create that instance.

So if you want to instantiate a class that is not registered, you have to do all these steps yourself. That's whatComLibrary is for. It has the following interface:

Hide Copy Code

```
class ComLibrary
{
  HRESULT Load(LPOLESTR szDllName);
  HRESULT Unload(BOOL bForce = FALSE);

  HRESULT GetClassObject(REFCLSID aClsid, REFIID aIid, LPVOID FAR* ppv);
  HRESULT CanUnloadNow();

  template<class Iface> HRESULT CreateInstance(
      REFCLSID aClsid,
      Iface** aRetPtrPtr,
      IUnknown* aUnknownOuter = nullptr);
};
```

Using it is straight forward, as you might guess from the interface. For each module you are planning to use, provide an instance of ComLibrary. It has to live as long as you are using that module:

Hide Copy Code

```
class MyApp
{
...
  private:
    ComLibrary  mMathModule;
}
```

Load the library on startup. Note that the path has to be absolute and can't be a network path to prevent security issues with loading DLLs from unexpected sources:

Hide Copy Code

```
HRESULT MyApp::Init()
{
  HRESULT hr = mMathModule.LoadModule(GetAppPath() + 'mathmodule.dll');
  if (FAILED(hr))
  {
    // error handling
  }
}
```

And then, instead of calling CoCreateInstance(...), do:

Hide Copy Code

```
...
IVector * pVector = nullptr;
```

```
HRESULT hr = mMathModule.CreateInstance(CLSID_Vector, &pVector);
// error checking etc.
```

You will still have to specify a `CLSID` of course. When you create a COM DLL, you probably use MIDL to define your interfaces. If so, the MIDL compiler will create a *_h.h* and a *_i.c* file. Include them in your application, and you have everything you need to use your interfaces and `CLSID`s.

When you are done using your instances, just release them as usual. If you feel you are done using your mathmodule, you can call `UnloadModule(...)`. This will check if there are still outstanding references to that module (via `DllCanUnloadNow()`) and unload the DLL if there are no more instances in use, or if you specify`bForce`. Usually, you would do this before exiting your application.

A few notices about the module DLLs: When you create a ATL COM DLL using the assistant from VS, you end up with a DLL that can be registered. If you don't want that, you can simply remove some stuff. This also helps to keep things clean. So what you should do is:

- Remove some resources, namely the *.rgs* files.
- Check your `CAtlDllModule` class (usually in *dllmain.h*) and remove the macro`DECLARE_REGISTRY_APPID_RESOURCEID`.
- In your `CAtlDllModule` implementation file, remove the `DllRegisterServer()`,`DllUnregisterServer` and `DllInstall` functions. Remove them also from your *.def* file.
- In all your COM classes, replace the macro `DECLARE_REGISTRY_RESOURCEID(...)` with`DECLARE_NO_REGISTRY()`

## Sample Code

The sample code uses ATL, although the `ComLibrary` class doesn't, you can use it in plain COM.

## License

## About the Author

### imagiro

Software Developer
Germany 🇩🇪
Born in 1968 I do programming since over 25 years now. I started with Basic on a ZX81 and with hacking hexcodes in a Microprofessor before I switched to C++ and other languages.

Since more than 10 years I work as a professional software developer, currently for Salsitasoft in Prague.