How do I obtain the computer manufacturer's name via C++?

devblogs.microsoft.com/oldnewthing/20140106-00

January 6, 2014

Raymond Chen

The way to get the computer manufacturer and other information is to ask WMI. WMI is much easier to use via scripting, but maybe you want to do it from C++. Fortunately, MSDN takes you through it step by step and even puts it together into a sample program.

But I'm going to write the code myself anyway.

Today's Little Program extracts the computer name, manufacturer, and model from WMI. Remember that Little Programs do little or no error checking.

And the smart pointer library we'll use is (rolls dice) `_com_ptr_t`!

```
#include <windows.h>
#include <stdio.h>
#include <ole2.h>
#include <oleauto.h>
#include <wbemidl.h>
#include <comdef.h>
_COM_SMARTPTR_TYPEDEF(IWbemLocator, __uuidof(IWbemLocator));
_COM_SMARTPTR_TYPEDEF(IWbemServices, __uuidof(IWbemServices));
_COM_SMARTPTR_TYPEDEF(IWbemClassObject, __uuidof(IWbemClassObject));
_COM_SMARTPTR_TYPEDEF(IEnumWbemClassObject, __uuidof(IEnumWbemClassObject));
// CCoInitialize class incorporated by reference
```

Those include files and macros set things up so we can use `_com_ptr_t` to access WBEM interfaces.

```
_bstr_t GetProperty(IWbemClassObject *pobj, PCWSTR pszProperty)
{
  _variant_t var;
  pobj->Get(pszProperty, 0, &var, nullptr, nullptr);
  return var;
}
void PrintProperty(IWbemClassObject *pobj, PCWSTR pszProperty)
{
 printf("%ls = %ls\n", pszProperty,
  static_cast<PWSTR>(GetProperty(pobj, pszProperty)));
}
```

The first helper function retrieves a string property from a WBEM object. The second one prints it. (Exercise: Why do we need the `static_cast`?)

```
int __cdecl main(int, char**)
{
 CCoInitialize init;
 IWbemLocatorPtr spLocator;
 CoCreateInstance(CLSID_WbemLocator, nullptr, CLSCTX_ALL,
  IID_PPV_ARGS(&spLocator));
 IWbemServicesPtr spServices;
 spLocator->ConnectServer(_bstr_t(L"root\\cimv2"),
  nullptr, nullptr, 0, 0, nullptr, nullptr, &spServices);
 CoSetProxyBlanket(spServices, RPC_C_AUTHN_DEFAULT,
  RPC_C_AUTHZ_DEFAULT, COLE_DEFAULT_PRINCIPAL,
  RPC_C_AUTHN_LEVEL_DEFAULT, RPC_C_IMP_LEVEL_IMPERSONATE,
  0, EOAC_NONE);
 IEnumWbemClassObjectPtr spEnum;
 spServices->ExecQuery(_bstr_t(L"WQL"),
  _bstr_t(L"select * from Win32_ComputerSystem"),
   WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
   nullptr, &spEnum);
 IWbemClassObjectPtr spObject;
 ULONG cActual;
 while (spEnum->Next(WBEM_INFINITE, 1, &spObject, &cActual)
                                    == WBEM_S_NO_ERROR) {
  PrintProperty(spObject, L"Name");
  PrintProperty(spObject, L"Manufacturer");
  PrintProperty(spObject, L"Model");
 }
 return 0;
}
```

And here is the actual guts of the program.

We initialize COM but we do not call `CoInitializeSecurity` because the checklist notes that the call sets the default security for the entire process, which would be a global solution to a local problem. Now, in this case, we are in control of the process, but I'm doing it this way because I know people are going to copy/paste the code (hopefully after adding some error checking), and the local solution is more appropriate in the general case.

The next step in the cookbook is creating a connection to a WMI namespace. We create a `WbemLocator` and connect it to the desired namespace.

Step three in the cookbook is setting the security context on the interface, which is done with the amusingly-named function `CoSetProxyBlanket`.

Once we have a connection to the server, we can ask it for all (`*`) the information from `Win32_ComputerSystem`.

We know that there is only one computer in the query, but I'm going to write a loop anyway, because somebody who copies this code may issue a query that contains multiple results.

For each object, we print its Name, Manufacturer, and Model.

And that's it.

Raymond Chen

Follow