Calling C++ code from Python

***Abstract.*** *This article explains how to write a C++ code in Visual Studio/Windows than can be called from Python. It uses the native* [*ctypes*](http://docs.python.org/library/ctypes.html) *Python module and bypasses other extensions such as SWIG and Boost.Python.*

Note: The method using ctypes will work on any python version, not just on the one it was compiled against as there are no compile time dependencies on python.

Python can import any C linkage function from a Windows DLL. This is declared using the **extern "C" \_\_declspec(dllexport)** directive.

1. Creating the DLL

In the following example we create a simple function that adds two integers. It is declared extern "C" to avoid the complicated C++ function name decoration in the object file and tell the compiler it should be exported from the dll by adding \_\_declspec(dllexport).

// in main.cpp

extern "C" \_\_declspec(dllexport) int sum(int a, int b)

{

return a + b;

}

The code should be added to a Dinamic-Link Library project and compiled. According to MSDN a (DLL) project can be created using the following steps:

1. On the menu bar, choose **File**, **New**, **Project**.
2. In the left pane of the **New Project** dialog box, expand **Installed**, **Templates**, **Visual C++**, and then select **Win32**.
3. In the center pane, select **Win32 Console Application**.
4. Specify a name for the project in the **Name** box. Specify a name for the solution **Solution name** box. Choose the **OK** button.
5. On the **Overview** page of the **Win32 Application Wizard** dialog box, choose the **Next** button.
6. On the **Application Settings** page, under **Application type**, select **DLL**.
7. Choose the **Finish** button to create the project.

We chose to name the library exportingtoPython.dll and make sure it is in the path.

1. Using the library from C/C++

First, the DLL should be loaded by calling LoadLibrary. This returns a handle that identifies the DLL in the current process.

Then the address of each function that the program will call should be associated to a local function pointer variable.

Note: Both the module handle and the function pointers are local the calling process (e.g. cannot be inherited in a multi-threading environment).

After these steps the function can be called in the usual way until the DLL is loaded in the process local address space.

int main()

{

HMODULE H = LoadLibrary("exportingtoPython.dll");

typedef int (\*funcPtr)(int, int);

funcPtr sum;

sum = (funcPtr) GetProcAddress(H, "sum");

cout << sum(5, 10) << endl;

FreeLibrary(H);

}

1. Using the DLL from Python

The [ctypes](http://docs.python.org/library/ctypes.html) module is used to access the DLL:

from ctypes import cdll

mydll = cdll.LoadLibrary('exportingtoPython.dll')

On **Windows**, cdll.LoadLibrary will search for the DLL in the current working directory.

The *sum* function from *mydll* can be accessed as

mydll.sum(5, 10)

Further work:

* See <https://docs.python.org/2/extending/extending.html> for error handling, exceptions, etc.