



## Application Note #4433

### Getting Started with Galil and C++ Programming

Programming the Galil controller from a high level language such as C or C++ can be overwhelming at first glance, but upon closer inspection – it is actually very straightforward. This article shows how to compile the “Hello World” type of application that is included in the GalilTools installation and then goes on to more advanced topics of how to create a new C++ Class specific to your motion control application<sup>1</sup>.

#### Part I – Compiling hello.cpp

Install GalilTools and connect to your controller to make sure you can communicate. From there, open up the Visual Studio C++ (2008) Command prompt and change your directory to be C:\Program Files\Galil\GalilTools\lib as shown here:

```

c:\ Visual Studio 2008 Command Prompt
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
C:\Program Files\Microsoft Visual Studio 9.0\VC>cd \
C:\>cd "Program Files\Galil\GalilTools\lib"
C:\Program Files\Galil\GalilTools\lib>
  
```

Make sure that you can compile the hello.cpp without any errors. To do that, type in `cl hello.cpp Galil1.lib -EHsc -MD`

The output should look something like this:

```

c:\ Select Visual Studio 2008 Command Prompt
C:\Program Files\Galil\GalilTools\lib>cl hello.cpp Galil1.lib -EHsc -MD
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

hello.cpp
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:hello.exe
hello.obj
Galil1.lib
C:\Program Files\Galil\GalilTools\lib>
  
```

Type in `hello` or `hello.exe`, it should bring up a GUI dialog to connect to a controller. Once you connect it should output a lot of information to the screen.

```

C:\ Visual Studio 2008 Command Prompt
185 _TI5, 255 , Digital inputs 41 to 48
186 _TPA, 0 counts, Axis A encoder position
187 _TPB, 0 counts, Axis B encoder position
188 _TPC, 0 counts, Axis C encoder position
189 _TTA, 0 U, Axis A torque <DAC>
190 _TTB, 0 U, Axis B torque <DAC>
191 _TTC, 0 U, Axis C torque <DAC>
192 _TUA, 0 counts/s, Axis A filtered velocity
193 _TUB, 0 counts/s, Axis B filtered velocity
194 _TUC, 0 counts/s, Axis C filtered velocity
195 _ZAA, 0 , Axis A user variable
196 _ZAB, 0 , Axis B user variable
197 _ZAC, 0 , Axis C user variable
2010 COMMAND ERROR. Galil::command("this is an error") got ? instead of : response. T
nd"
a command error occurred
C:\Program Files\Galil\GalilTools\lib>

```

Now you've successfully compiled a program in C++ that communicates to the Galil controller. A basic C++ program can be accomplished quite easily by simply using the hello.cpp as a guide and modifying it to send commands back and forth to the controller. The hello.cpp file should be reviewed carefully as it provides an example of each of the GalilTools COM api functions. These include uploading/downloading programs & arrays, and working with the controllers data record.

## Part II (Advanced) – Creating a Custom C++ Class

For the second part of this article, we are going to create a new class that allows the user to expand the basic GalilTools api. The following assumes that you are familiar with C++ programming – if you are not, don't worry there are lots of good C++ programming guides – and you don't have to be an expert, just knowledgeable in the basics.

A "class" is similar to a data structure (struct) except that it can hold both data and functions – both of which are considered "members" of the class. Once you have defined the definition of your class, you then create an "object" that holds the information at runtime. You can create multiple objects in your application (aka: object oriented programming). The example below creates a "Controller" class and then uses the object to send/receive commands to the controller.

First, create new directory in the lib folder of GalilTools and name it cpp-demo. Copy Galil1.dll, Galil1.lib, Galil.h into the directory. Create a new file called dmc-class.h with the following code:

```

//dmc-class.h
#include "Galil.h"
#include <string>

class Controller {
public:
    Controller(const std::string& address = ""); //default to empty string
    double Controller::Reference_Position(std::string& axis);
    double Abort(double value);

~Controller();
private:
    Galil *g; //must be a pointer to g instead of g
    int axes;

```

```
};
```

The .h file above is the header file and lists all of the members of the class without going into the actual details of what they do. Every member needs to be listed in the header file and then created in the .cpp file. At this point, we know that there is a new class called Controller with a function called Reference\_Position that takes a string as an argument and returns a “double” integer value. Likewise, there is an Abort function that takes a value as an input and returns a double. These are just two example functions to show how functions work – you will need to create more that are specific to your needs.

So, now let’s make a new file called dmc-class.cpp and put in the following code:

```
//dmc-class.cpp
#include "Galil.h"
#include "dmc-class.h"
#include <iostream> //cout
#include <sstream> //stringstream
using namespace std;

Controller::Controller(const std::string& address) {
    cout << "Creating Controller with address " << address << "...\\n";
    g= new Galil(address); //this creates the Galil object
}

///
```

Here we can see the details of our new class. The Controller::Controller is called the constructor and runs anytime a new Controller object is created. The string that is passed is the address, a new connection is created to the Galil with the standard GalilTools api - it is called “g”. Next there are the two functions – the first of which will send down the command RP followed by the string that was passed to the function. It returns the commanded position of the axis. The second function is similar in that it sends down the “AB” command. Lastly a destructor is created that occurs when the object is killed.

It is important to note that at this point, we have not written the actual program yet. We've just set up a class that will allow us to write the program in a manner that makes coding easier. So, now we can write the code that will cause something to happen. Create a new file called test-dmc-class.cpp and copy in the following code:

```
//cl test-dmc-class.cpp dmc-class.cpp Galil1.lib -EHsc -MD

#include "Galil.h" //vector string Galil
#include "dmc-class.h"
#include <iostream> //cout
#include <sstream> //ostringstream istream
using namespace std; //cout ostream vector string

int main()
{
    try
    {
        Controller DMC(""); //supply address string or empty for dialog
        string A ="A";
        cout << DMC.Reference_Position(A) << endl;
        DMC.Abort(1); //abort motion
    }

    catch(string e) //error
    {
        cout << e;
        if(string::npos != e.find("COMMAND ERROR"))
            cout << "a command error occurred"; //catch command errors
        return 1; //error. 0 is normal.
    }
} //main
```

This program simply sends down a command that requests the A axis reference position and then sends the Abort command. However, the benefit is that you have done the dirty work of string manipulation in the class file so the code here is clean and easy to read. It also allows you to re-use code and prevents bugs due to syntax errors. If you are using a Visual Studio IDE, it has the added benefit of intellisense auto-completion of your class functions. A Class can also be used to put max/min limits on input values, convert from real world units (mm,inch) to counts, and much more.

Compile the code with the following command line function:

```
cl test-dmc-class.cpp dmc-class.cpp Galil1.lib -EHsc -MD
```

the output is shown below:

```

Visual Studio 2008 Command Prompt

C:\Program Files\Galil\GalilTools\lib\cpp-demo>cl test-dmc-class.cpp dmc-class.c
pp Galil1.lib -EHsc -MD
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

test-dmc-class.cpp
dmc-class.cpp
Generating Code...
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

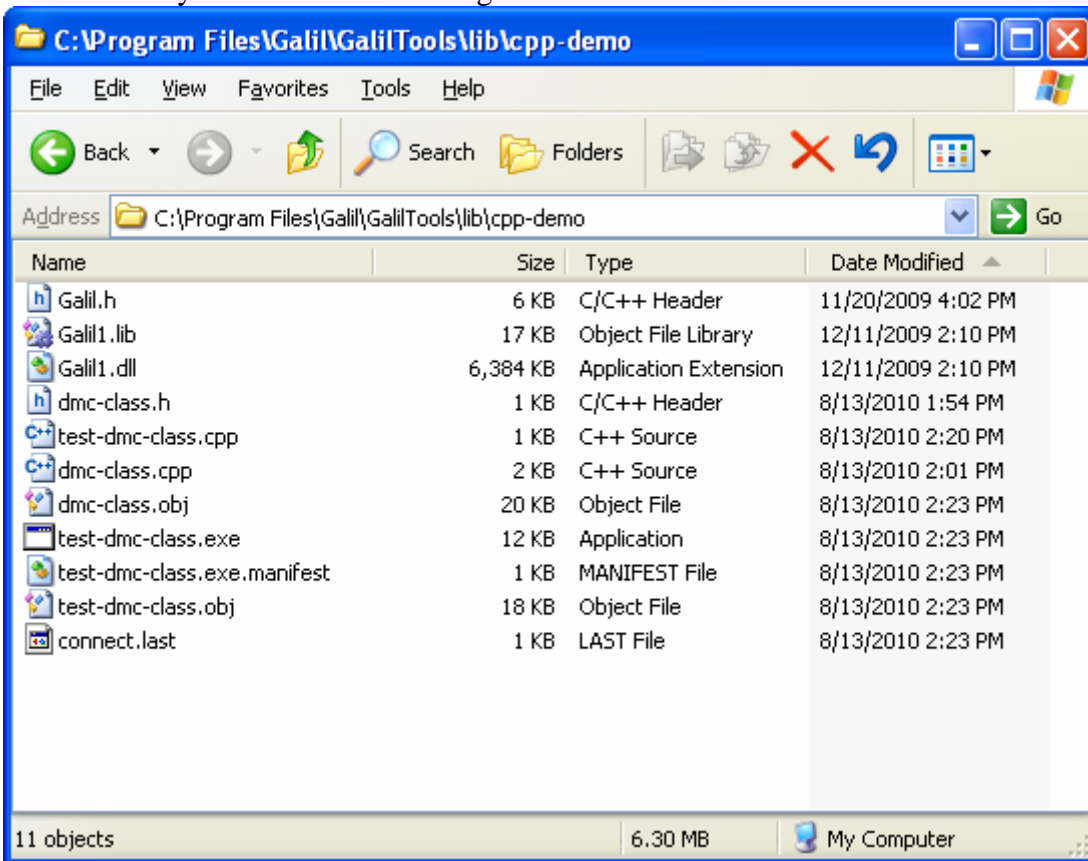
/out:test-dmc-class.exe
test-dmc-class.obj
dmc-class.obj
Galil1.lib

C:\Program Files\Galil\GalilTools\lib\cpp-demo>test-dmc-class
Creating Controller with address ...
0

C:\Program Files\Galil\GalilTools\lib\cpp-demo>

```

Your directory should look something like this:



## **Deployment**

Once you have created your C++ application, you will need to make sure that you copy the correct files over to the target PC. The following application note shows exactly what files you need to deploy your application:

<http://www.galilmc.com/support/appnotes/software/note4432.pdf>

## **Conclusion**

Galil has provided the hooks needed to get up and running quickly whether you have just a simple application or something much more complex. Programming in a C/C++ environment can have rewarding benefits of speed and cross-platform portability.

Although, this example was done using the command line, the same exact practice holds true for GUI applications and when using the Visual Studio IDE.

## **Notes**

<sup>1</sup> C# users can go to the following link for creating custom classes in .NET:

<http://www.galilmc.com/techtalk/software/how-to-create-a-custom-class-library-with-galiltools/>