**Multiplayer Assembly Package  
Tutorials  
Creating a new Class to work with MAP**

This tutorial will teach you how to use the MAP control header file to create new classes that are dependent on the systems found within the pack.

At some point, you will likely want to create a new container of some form, or to have something hold data to be encrypted later. This pack includes the Crypto++ Library, and a vast array of easy to use cryptographic (think encryption) methods. This tutorial will show you how I have constructed my classes to work with Torque 3D and how you can use the MAP to toggle them on or off depending on your specific project.

This is a relatively simplistic task to do, and it works a bit like this. Create a new header file. Name it accordingly and place it in the solutions folder of the PGD folder. Add the following to it:

#ifndef my\_new\_class\_H

#define my\_new\_class\_H

#ifndef \_CONSOLE\_H\_

#include "console/console.h"

#include "console/consoleInternal.h"

#endif

class myNewClass {

public:

static void create();

static void destroy();

myNewClass();

~myNewClass();

private:

};

#endif

extern myNewClass \*mein;

Replace my\_new\_class and myNewClass accordingly, also be sure the extern pointer to your class is named with a unique identifier, you will get a C++ error on compiling if this is not the case. You do not need to include the CONSOLE\_H section if your new class does not reference the console in any way. I just find it good practice to include it anyways.

Now we will create our C++ file. Do so and give it the same name as your header file but with the .cpp extension. Add the following contents to it:

#include "PGD/Solutions/myNewClass.h"

myNewClass \*mein = NULL;

void myNewClass::create() {

if(mein == NULL) {

mein = new myNewClass();

Con::printf("MAP Module Loaded: My Class");

}

}

void myNewClass::destroy() {

if(mein != NULL) {

delete mein;

mein = NULL;

Con::printf("MAP Module Removed: My Class");

}

}

myNewClass::myNewClass() {

//init variables here

}

myNewClass::~myNewClass() {

}

Obviously make the same changes as you did earlier with the naming, and ensure you are calling the correct header file above. This is a great template to starting your class out, and I recommend always beginning here when creating a new class. Now let’s get T3D to recognise it.

Open the file PGD/Control/PGDMain.h

Now a bit on how this file works. You will see a block of code under \_LOADALL with multiple header file calls, add your file to this list. Basically the \_LOADALL definition is used to load all of the header files in that list when necessary. Below is the definition for each individual game project you may have. The default one is \_PGD\_MYGAME. This is defined at the top of the file to control what project loads are to be used, very useful when deploying MAP to multiple projects. Inside the define block you will see a format of different loads. Bascially how this section works is a file calls this definition and then the PGDMain.h file recognises the block and loads the reference file.

So, create a block of this style under the \_PGD\_MYGAME block:

#ifdef \_LOADPGDMYCLASS

#include "PGD/Solutions/myNewClass.h"

#endif

Change the file call and the MYCLASS a necessary. Now we need to tell our game to load our new class, so open up the file: app/mainLoop.cpp

You may recall that in the installation of this pack we had to make some simple modifications to this file. Scroll down to where you added the first block of code and the include line: #include "PGD/Control/PGDMain.h". Above this you will see the define statements for every individual part of the pack. At some point I will change this to use \_LOADALL but for now we need to load them individually. Above the PGDMain.h definition add the line: #define \_LOADPGDMYCLASS 1

Now scroll down to the standardMainLoop::init() modifications and you will see a similar setup. Add your new class as follows:

#ifdef \_LOADPGDMYCLASS

myNewClass::create();

#endif

Make all modifications as necessary, then save all files and build your solution. When you run your game, you should see the console message for class creation appear in your list, if this is the case, you’re all done!

A little post tutorial note: This method even works when creating a new console controlled class for just plain old T3D, just augment the #ifdef segment when creating the class in ::init() and load your header file as you normally would.