**Multiplayer Assembly Package  
Tutorials  
Creating a Client-Side Rank/Progression System**

This tutorial will teach you how to create a client-side Rank/Progression system using the Multiplayer Assembly Package.

**What is it?**A Client-Side Rank/Progression system is mainly used in single-player games and RPG games, such as The Elder Scrolls Series, or Borderlands. I use this method for my PGD Survival game. The effectiveness of this system is the fact that no transmissions to the server are needed and that all of the necessary information is stored on the client. Obvious downfalls include loss of data if a hard drive corruption occurs and potential hacks if the data is not secured. Thankfully, this pack will protect you from case #2. I include a bit on how to protect from Case #1 at the end of this tutorial, but a more in-depth example would be to look at the Server-Side Rank Progression Tutorial.

**Model**Let’s talk about what we need to set this system up. I will consider a FPS rank-progression system similar to that of CoD. First we need to define a table of experience values. There are two easy ways to think of doing this. We can either have a big numeric “Experience” field that contains the player’s overall experience and the table will have value at which point the next level is attained, or we can approach this via “Remaining\_Experience” in which case the value of this field decreases to zero and then a level counter goes up by one. In this tutorial, I will teach you how to design and implement an incremental system.

Also, In more recent CoD games, they also offered a feature in which a player may reset his rank back at 0 to increase a “prestige” level, which is simply another numeric value stored somewhere. I’ll have a little bit on that at the end.

***C++ Code Work***Personally, I like to work with new files when designing internal systems. If you have not yet read my tutorial on creating new T3D classes to work with MAP, please read this now as I will simply write the code for this container and explain it.

To create an incremental experience container, we will use the following code. Create a new file named experienceSystem.cpp, and place it in the PGD/Solutions folder

#include "PGD/Solutions/experienceSystem.h"

#include "platform/platform.h"

#include "console/console.h"

#include "console/consoleInternal.h"

#include "console/engineAPI.h"

using namespace std;

expControl \*experience = NULL;

const F32 expControl::expTable[MAX\_LEVEL] = {0, 50, 100, 150, 200, 250, 300};

//ASSET FUNCTIONS

const char \* expControl::F32ToConstChar(F32 in) {

char output[32];

dSprintf(output, 32, "%.0f", in);

return output;

}

const char \* expControl::intToConstChar(int in) {

char output[sizeof(int)];

dSprintf(output, sizeof(output), "%i", in);

return output;

}

expControl::expControl() {

}

expControl::~expControl() {

}

void expControl::create() {

if(experience == NULL) {

experience = new expControl();

Con::printf("Expereince System Online");

}

}

void expControl::destroy() {

if(experience != NULL) {

delete experience;

experience = NULL;

}

}

F32 expControl::getEXP() {

return plEXP;

}

F32 expControl::getNeededEXP() {

int level = getLevel();

if(level >= MAX\_LEVEL) {

return getEXP();

}

return expTable[level];

}

int expControl::getLevel() {

F32 exp = getEXP();

//special cases

if(exp >= expTable[MAX\_LEVEL - 1]) {

return MAX\_LEVEL;

}

//

for(int i = 0; i < MAX\_LEVEL; i++) {

if(exp >= expTable[i]) {

//need more

}

else {

return i;

}

}

return -1;

}

void expControl::addEXP(F32 amount) {

F32 prior = getEXP();

int currentLV = getLevel();

plEXP += amount;

if(plEXP >= expTable[currentLV]) {

Con::evaluatef("onLevelUp(%i);", currentLV+1);

//check for additional levels

if(plEXP >= expTable[currentLV+1]) {

//loop through the remaining ones

for(int i = currentLV+1; i < MAX\_LEVEL; i++) {

if(plEXP >= expTable[i]) {

Con::evaluatef("onLevelUp(%i);", currentLV+1);

}

}

}

}

}

//File Controls

void expControl::saveInfo(const char \*guid, const char \*file) {

FileObject \*fileObj = new FileObject();

fileObj->registerObject();

fileObj->openForWrite(file, false);

String lineWrite;

Con::printf("SaveInfo: %s => %s", guid, file);

//write the XML file Heading

fileObj->writeLine((const U8 \*)"<?xml version=\"1.0\" encoding=\"utf-8\" standalone=\"yes\" ?>");

lineWrite = "<!-- Saved Data for: ";

lineWrite += guid;

lineWrite += " -->";

fileObj->writeLine((const U8 \*)lineWrite.c\_str());

fileObj->writeLine((const U8 \*)"<Data>");

lineWrite = " <Info value=\"exp\">";

lineWrite += F32ToConstChar(plEXP);

lineWrite += "</Info>";

fileObj->writeLine((const U8 \*)lineWrite.c\_str());

fileObj->writeLine((const U8 \*)"</Data>");

fileObj->close();

fileObj->destroySelf();

//encrypt the file using AES

cryptoPackage->AESEnc\_File(cryptoPackage->AESFileKey(), file, 1024);

}

void expControl::loadInfo(const char \*file) {

SimXMLDocument \*xml = new SimXMLDocument();

if(!Platform::isFile(file)) {

Con::errorf("loadInfo: File %s not found.", file);

return;

}

xml->registerObject();

cryptoPackage->AESDec\_File(cryptoPackage->AESFileKey(), file, 1024);

xml->loadFile(file);

cryptoPackage->AESEnc\_File(cryptoPackage->AESFileKey(), file, 1024);

xml->pushChildElement(0);

xml->pushFirstChildElement("Data");

while(true) {

xml->pushFirstChildElement("Info");

while(true) {

//set the attributal info

if(strcmp(xml->attribute("value"), "exp") == 0) {

const char \*exp = xml->getData();

plEXP = dAtof(exp);

}

else {

Con::errorf("FILE LOAD ERROR: Invalid Field %s found in weapon load", xml->attribute("value"));

}

//check for next item.

if(!xml->nextSiblingElement("Info")) {

break;

}

}

xml->popElement();

if(!xml->nextSiblingElement("Data")) {

break;

}

}

xml->destroySelf();

}

Now create a header file named experienceSystem.h and place it in the same directory as the CPP file. Add the following to it:

#ifndef \_CONSOLE\_H\_

#include "console/console.h"

#include "console/consoleInternal.h"

#endif

#include "PGD/Crypto/cryptoPackage.h"

#include "PGD/Solutions/PGDStore.h"

#include "math/mRandom.h"

#include "console/SimXMLDocument.h"

#include "core/fileObject.h"

#define MAX\_LEVEL 7

#ifndef experienceSystem\_H

#define experienceSystem\_H

class expControl {

private:

F32 plEXP;

public:

const char \*F32ToConstChar(F32 in);

const char \*intToConstChar(int in);

static const F32 expTable[MAX\_LEVEL];

static void create();

static void destroy();

expControl();

~expControl();

void addEXP(F32 amount);

F32 getEXP();

F32 getNeededEXP();

int getLevel();

//Control Functions

void saveInfo(const char \*guid, const char \*file);

void loadInfo(const char \*file);

};

#endif

extern expControl \*experience;

Follow the new class creation tutorial to add this new module to your list of classes. Now, let’s talk about how this works.

Some of the obvious stuff first. Inside the header file is a MAX\_LEVEL definition. This is the amount of levels your player will be able to obtain in your game. This number MUST be the number of array elements inside the expTable array.

You should also notice some calls to the cryptography system in here, these are mainly in the file save and load commands which will protect your client-side data from evil script majik, just don’t be the silly developer who gives console access to AES File Decryption, otherwise you can kiss that goodbye.

Now, let’s discuss the TorqueScript side of things.

First off, we will need to define some new console functions, these will give your game access to this rank container. At the bottom of the C++ file you created, add the following:

DefineEngineFunction(addExperience, void, (F32 amount),, "(F32)") {

experience->addEXP(amount);

}

DefineEngineFunction(saveClientData, void, (const char \*guid, const char \*path),, "(string, string) save data") {

if(strcmp(guid, "") == 0) {

Con::errorf("saveClientData() - error, cannot save without a valid guid.");

return;

}

if(strcmp(path, "") == 0) {

Con::errorf("saveClientData() - error, cannot save without a valid path.");

return;

}

experience->saveInfo(guid, path);

}

DefineEngineFunction(loadClientData, void, (const char \*path),, "(string) load data") {

if(strcmp(path, "") == 0) {

Con::errorf("loadClientData() - error, cannot save without a valid path.");

return;

}

experience->loadInfo(path);

}

DefineEngineFunction(getInfo, const char \*, (),, "() push the info string") {

char exp[32], needed[32], level[4];

dSprintf(exp, sizeof(exp), "%.0f", experience->getEXP());

dSprintf(needed, sizeof(needed), "%.0f", experience->getNeededEXP());

dSprintf(level, sizeof(level), "%i", experience->getLevel());

String output;

output += exp;

output += "\t";

output += needed;

output += "\t";

output += level;

return output.c\_str();

}

This will add the console functions we need to the game. Now let’s explain how to set up your system correctly. Create a file or place this in a file on the client side:

What you want to do is write a clientCmd that accepts values from the server. The server will then transmit this message to the client when they perform an action worthy of gaining EXP.

IE:

function clientCmdGainEXP(%amount) {

addExperience(%amount);

}

Now, on the server side of things, you would write a function in onDeath() that calls this client CMD:

commandToClient(%killerClient, ‘GainEXP’, 5);

Replacing where necessary. This example would give %killerClient ‘5’ EXP for killing an enemy player. You woild obviously code the checks for game mode, and EXP scoring, but this point is rather simple enough to understand…

To use the save and load functions, I recommend calling them the instant your client signs into your game. See client/Auth/GuiWindows.cs there is a function for after log-in is completed. You would call the load function here (if the file exists, besure to check for that first). Simply use the $ConnStore::Guid variable to obtain the client GUID. For the first time (if the file does not exist), directly call the save function as it will create the XML file with the value of 0 EXP (as set by the class).

Other than that, this system is quite simplistic and self explanitory. Now, onto the post-turotial stuff.

**Post Tutorial Stuff**

I mentioned at the beginning of the tutorial that I would give a piece on the CoD prestige system and how it works. Basically, this class you created now functions as a information container (like the PGDStore class I included with MAP). To create a prestige system, you would first do: #define MAX\_PRESTIGE\_LEVEL x, replacing x with whatever you want. Then, you would add a int field to the private section containing the player’s prestige rank.

To use this field, you would write a new C++ funtion that checks the current EXP of the player. If he is at the max level, you would set the player EXP to 0, and add one to the new prestige level, and then immediately call the save function. Add this function to the class and then defineEngineMethod your new function so TS may use it.

To add the prestige level to the saver, you would model it after my EXP save with a <Info value=”prestige”>#</Info> set up. You can see the Torque 3D SimXMLDocument doccumentation if you need more help with the formatting of the C++ syntax.

Lastly be sure to modify the getInfo TS call to include the prestige level so your GUI screens can use them.

One last piece. You might notice in the addExperience function that a call to “onLevelUp” is made. This will be called when your player levels up to level %i, so feel free to make a new GUI to reflect this.

For other GUIs that you want to display the level info simply do a: %info = getInfo(); and then you can access the information like so:

%exp = getField(%info, 0);

%neededEXP = getField(%info, 1);

%level = getField(%info, 2);

Good luck, and enjoy your new RP system!