CH3_001

EXERCISE

NAMED CONNECTIONS

Exercise Files

Starter – "Kit/gpgt/server/scripts/gpgt/chapter3/exercise001.cs"
Answers – "Kit/gpgt/server/scripts/gpgt/chapter3/answers/exercise001_f.cs"

Exercise Mission

Chapter 3: "001_GameConnection: Named Connections"

Synopsis

Any time a standard Torque game utilizes the engine in client-server mode, two named connections are created, localClientConnection and serverConnection. These connections are named in this fashion to simplify our lives while we design, test, and debug our games. This exercise focuses on familiarizing you with these two named connections.

Prerequisites

1. *ch1_001.pdf "Using The Kit"*

Exercises

1. *Examining localClientConnection (pg 2)*
2. *Examining serverConnection (pg 4)*

NAMED CONNECTIONS

1 Examining localClientConnection

First, let us examine localClientConnection. If you will recall, this instance of GameConnection is automatically created for us when a client connects to the server using an internal connection. In other words, a single-player game using the single-player connection and a multi-player game where we are using the listen-server connection (and playing on the engine running in client-server mode), will both automatically create and name this instance of GameConnection.

To see this, please do the following.

1. In the current mission, open the console (~).
2. Type the following code.

```
cls();
localClientConnection.dump();
```

This will clear the console (making it easier to review the results of the next command), and then list all of the field values for localClientConnection, as well as all the methods associated with it.

```
==>cls();
==>localClientConnection.dump();
Member Fields:
  canSaveDynamicFields = "1"
Tagged Fields:
  armor = "Light"
  Camera = "1620"
  currentPhase = "3"
  gender = "Male"
  guid = "0"
  isAdmin = "1"
  isSuperAdmin = "1"
  name = "\c011"
  nameBase = "Fresh Meat"
  Player = "1621"
  race = "Human"
  score = "0"
  sendGuid = "0"
  skin = "\c010"
Methods:
  activateGhosting() -
  add() - set.add(obj1,...)
  bringToFront() - set.bringToFront(object)
  chaseCam() - (int size)

... etc.
```


NAMED CONNECTIONS

Camera and Player Fields

If you look closely at this list, you will see two interesting fields, camera and player. It is worth noting, that the standard scripts found in the "~/server/scripts/game.cs" automatically store the ID of the client's player and camera server-objects in every client connection. Since we have the name of the client connection, we can directly access this information.

Experimenting With Methods

Now, take a look at the long list of methods associated with localClientConnection. Many of them are probably unfamiliar, or uninteresting, but a few of them should pop-out after having read chapter 3.

- getAddress()
- getCameraObject()
- getPing()
- getServerConnection()
- setFirstPerson()
- ...

Try some of the following commands in the console.

```
localClientConnection.setFirstPerson(1);
echo( localClientConnection.camera );
echo( localClientConnection.player );
echo( localClientConnection.getCameraObject() );
```

The first command forces us into first POV. The second and third commands print the IDs of the camera and the player to the console. The last command prints the ID of the object being used for camera configuration. Which object is it?

Hmmm... now try this.

```
localClientConnection.setFirstPerson(0);
echo( localClientConnection.getCameraObject() );
```

Are you surprised that the same object is configuring the camera? You shouldn't be. We're still using the player as our camera object.

Now, try this.

```
localClientConnection.setCameraObject(  localClientConnection.camera );
echo( localClientConnection.getCameraObject() );
```

See that it has switched?

Now, try some other methods like getPing() or getAddress().

Do you see how this can be useful?

I think you will.

At this time, please exit the mission and re-start it, then proceed with the rest of this exercise.

2 Examining serverConnection

serverConnection is present in every client's copy of the engine. In fact, the only time it isn't present is when the engine is running in dedicated-server mode. Of course, that mode is reserved for servers only, so it makes perfect sense that there is no connection to a server in it.

The fact that this named connection is omnipresent in client game copies makes it useful for debugging, and for writing scripts in your games.

For example, we could try some of the same experiments we tried with clientConnection.

See what fields and methods are associated with this connection.

```
cls();
serverConnection.dump();
```

Switch the server to first POV.

```
serverConnection.setFirstPerson(1);
```

Switch the server to third POV.

```
serverConnection.setFirstPerson(0);
```

Get the address of the server.

```
echo(serverConnection.getAddress());
```

Locating serverConnection In The Scripts

This is probably all very interesting and useful, but there is something else you may want to do at some time in the future. You may want to locate the place where this connection is created and use a different class than GameConnection to create it.

Why?

Well, perhaps you want to create an AI-controlled client using the class AIConnection, or perhaps you want to create your own class derived from GameConnection and use it instead.

So, how do we find the places where serverConnection is created?

Easily.

NAMED CONNECTIONS

In your favorite editor, you simply need to search for this pattern "new GameConnection". You might be surprised, but this code is found in 5 places in the Kit.

Now, as an exercise, please match these uses:

1	Used to play back a demo.
2	Used for a single-player or a multi-player game where this player is hosting.
3	Used to connect to a remote server.
4	Used for a single-player game.

, to these "new GameConnection" locations:

File	Line #	Use
Kit\common\client\missionDownload.cs	126	
Kit\common\client\missionDownload.cs	38	
Kit\creator\editor\editor.cs	104	
Kit\gpgt\client\ui\joinServerGui.gui	350	
Kit\gpgt\client\ui\startMissionGui.gui	160	

