

CH4_001

EXERCISE

SERVER QUERIES

Exercise Files

Starter – "Kit/gpgt/server/scripts/gpgt/chapter4/exercise001.cs"

Answers – "Kit/gpgt/server/scripts/gpgt/chapter4/answers/exercise001_f.cs"

Exercise Mission

Chapter 4: "001_Servers: Server Exercises"

Synopsis

In this exercise we learn to make various kinds of server queries and how to run a master server.

Prerequisites

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

Exercises

1. *Setting Up (pg 2)*
2. *LAN Server Queries (pg 5)*
3. *Set Up Master Servers List (pg 9)*
4. *Running A Master Server (pg 11)*
5. *Master Servers Query (pg 14)*

SERVER QUERIES

1 Setting Up

Goal: Write scripts to configure and execute LAN and Master Server queries.

Starter Code: You are provided with an exercise package (exercisePackage_001) that includes the following function bodies.

- function initClientQuerySettings()
- function dumpClientQuerySettings()
- function dumpServerSettings()
- function onServerInfoQuery()

Client Query Settings

In this part of this exercise, I want you to examine two of the supplied functions, initClientQuerySettings() and dumpClientQuerySettings().

initClientQuerySettings()

The initClientQuerySettings() function is provided to initialize our LAN and master server queries. Having this function will allow us to reset our queries to a known state while we experiment.

As you can see by examining the exercise starter file, I have supplied you with a function body and a set of variables already.

```
function initClientQuerySettings()  
{  
  //  $Client::LanPortQuery      =  
  //  $Client::FlagsQuery       =  
  //  $Client::GameTypeQuery    =  
  //  $Client::MissionTypeQuery =  
  //  $Client::MinPlayersQuery  =  
  //  $Client::MaxPlayersQuery  =  
  //  $Client::MaxBotsQuery     =  
  //  $Client::RegionMaskQuery  =  
  //  $Client::MaxPingQuery     =  
  //  $Client::MinCPUQuery      =  
  //  $Client::FilterFlagsQuery =  
}
```

Your job is to uncomment the supplied code and to provide some good values for each of the query settings. Please do so now.

Hints:

1. Please refer to chapter for good values.

SERVER QUERIES

dumpClientQuerySettings()

We will use the `dumpClientQuerySettings()` function to dump out the current query settings. You don't really need a function like this, but for this exercise it is nice as a debug tool.

I have supplied the beginnings of this function. Please fill in the rest of the statements now.

```
function dumpClientQuerySettings()
{
    echo("$Client::LanPortQuery    == ", $Client::LanPortQuery);
    // ... add remainder
}
```

Testing Your Functions

Now, let's test our new client query settings functions. Please follow these steps.

1. If you still have the kit running, exit to the main menu; otherwise, start the kit.
2. Start the mission for this exercise.
3. Open the console (~) and type the following.

```
initClientQuerySettings();
dumpClientQuerySettings();
```

What do you see?

Next, try modifying some settings and dumping them.

```
$Client::GameTypeQuery = "My awesome game";
$Client::MaxPlayersQuery = 9999;
dumpClientQuerySettings();
```

What do you see?

Lastly, try any additional settings you want to experiment with. Then, when you're satisfied, exit the mission (to main menu), and go on to the next part of this exercise.

Server Settings

As we learned in chapter 4, server settings (globals and the `onServerInfoQuery()` callback) provide important information to LAN and master server queries.

In this part of the exercise, we will create a function to dump some useful server settings and we will implement a basic `onServerInfoQuery()` callback.

SERVER QUERIES

dumpServerSettings()

As I just mentioned, the purpose of our dump server settings function is to print out all of the important server preferences and other globals that are used in setting up a server and are provided to queries. Chapter 4 lists these globals in several places, so they shouldn't be too hard to find.

Your job here is to fill in the body of the supplied function. (I've given you a hint on where to start.)

```
function dumpServerSettings()
{
    echo("$Server::Status          == ", $Server::Status );
    // ... add remainder

}
```

onServerInfoQuery()

Because this exercise uses the TorqueScript package feature, we can create new functions and methods that will override existing ones when the package is activated. So, I would like you to take the template I've given you in the exercise starter and to implement your own version of onServerInfoQuery(). Be creative or keep it simple. The decision is up to you.

```
function onServerInfoQuery()
{
}
```

Testing Your Functions

Now, let's test our new functions. Please follow these steps.

1. If you still have the kit running, exit to the main menu, otherwise start the kit.
2. Start the mission for this exercise.
3. Open the console (~) and type the following.

```
dumpServerSettings();
```

What do you see?

Next, try modifying some settings and dumping them.

```
$Server::Status = "Working fine";
$Server::Name = "My super server";
$Pref::Server::Info = "Home of a future AAA game.";
dumpServerSettings();
```

What do you see?

Lastly, try any additional settings you want to experiment with. Then, when you're satisfied, exit the mission (to main menu), and go on to the next part of this exercise.

SERVER QUERIES

2 LAN Server Queries

Goal: Learn about LAN Server Queries via experimentation.

Starter Code: You are provided with an exercise package (exercisePackage_001) that includes the following function bodies.

- function doLanServersQuery()
- function dumpServerInfoData()

Now that we've finished writing our setup functions, we can move on to something a little more interesting. Here, we will write two new functions, doLanServersQuery() and dumpServerInfoData().

doLanServersQuery()

This function will call the queryLANServer() console function supplied by Torque. In our implementation of the doLanServersQuery(), we will pass the client query values (from above) to queryLANServer.

Please fill in the supplied function body with a call to doLanServersQuery() using the proper client query variables (which we set up above).

```
function doLanServersQuery()
{
}
```

dumpServerInfoData()

Now that we have the query written, let's make a function to dump out all of the \$ServerInfo::* data which is generated by a query. This is a bit more complicated (not much), but I've supplied a function skeleton with the basic logic outlined.

Please fill in the supplied function body with the code necessary to count all servers that were found in the query and to subsequently dump the values of all \$ServerInfo::* variables to the console.

```
function dumpServerInfoData()
{
    // Find out how many servers were found.
    %count = 0 /*add correct function here */;

    echo("Found ", %count , " servers." );

    // Print out all of the server data we discovered
    for (%i = 0; %i < %count; %i++) {
        echo("Server # ", %i );

        echo("$ServerInfo::Status      == ", $ServerInfo::Status );
        // ... add remainder
    }
}
```

SERVER QUERIES

Bonus Exercise – Convert `$ServerInfo::Status` bitmask to strings

As we learned in chapter 4, the `$ServerInfo::Status` global will contain a bit mask representing the status of this query result record. Generally it is ignored, but if you poke around the engine code (C++), you will discover what the bit settings are and for this field.

- No bits set – This is a new record.
- bit 2 – Server is running Linux.
- bit 28 – Currently updating the record via call to `querySingleServer()`.
- bit 29 – Server responded with results. Valid record.
- bit 30 – Server query timed out. Invalid Record.

In this exercise, try writing code to cover this bitmask into a string.

See the finished code above for an example of how to do this.

Testing Your Functions

Now, let's test our new functions. Please follow these steps.

1. Be sure you deleted your preferences files (`prefs.cs`) and DSOs (`*.dso`).
2. If you still have the kit running, exit to the main menu, otherwise start the kit.
3. Start the mission for this exercise.
4. Open the console (`~`) and type the following.

```
doLanServersQuery();  
dumpServerInfoData();
```

What happened? You might have been expecting to see a server listed, but unless you had the “Host Multiplayer” checkbox checked, the server didn't advertise itself on the LAN. See figure 4.1 below to see what I mean by “selecting the checkbox”.

SERVER QUERIES

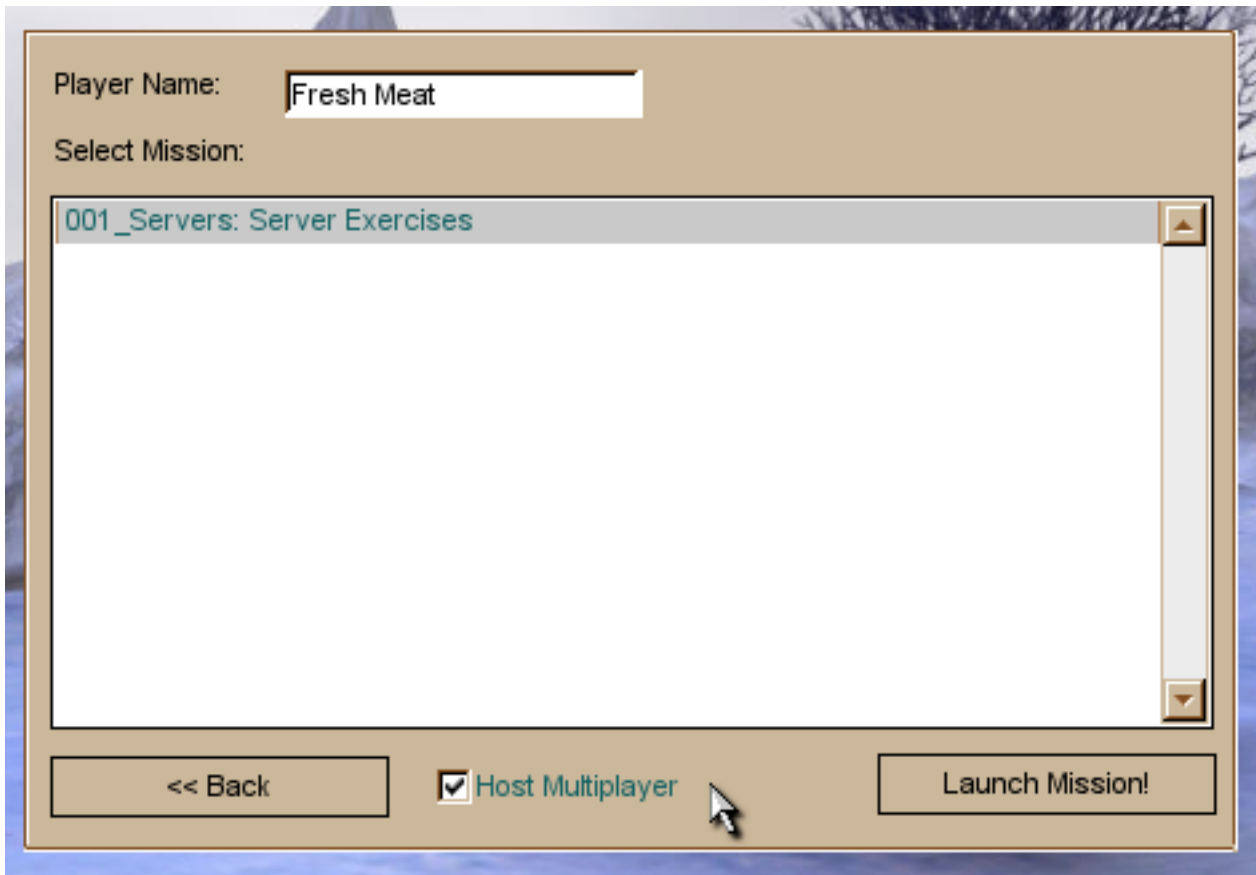


FIGURE 4.1 HOST MULTIPLAYER CHECKBOX SELECTED.

Now, let's try again, but with the checkbox selected.

1. Quit to the main menu.
2. Run the mission for this exercise and be sure “Host Multiplayer” is selected.
3. Open the console (~) and type the following.

```
doLanServersQuery();  
dumpServerInfoData();
```

What do you see?

SERVER QUERIES

Capture your IP

Before you move on, please write down the value that printed out for `$ServerInfo::Address`. When writing this and testing the code, my address looked like this:

```
$ServerInfo::Address      == IP:192.168.123.3:28000
```

The part of the address that we care about is “192.168.123.3”. So, write yours down and keep it for the next steps in our exercise.

My IP Address: _____

SERVER QUERIES

3 Set Up Master Servers List

Great. So far, we've gotten some important globals set up (for the exercise) and we've successfully run a LAN query. Now, we need to prepare to use a Master server.

If you recall, in order to advertise our game on a master server, we need to initialize the global(s) `$pref::Master[n]`, where `n` starts at 0 and is incremented for ever new master server we wish to advertise with.

In this step of the exercise, we will modify our master servers list and then write a function to dump out the master server address(es) to the console.

Modifying Our Master Servers List

In order to modify our master servers list, we need to do the following steps.

1. Stop the KIT, completely exiting.
2. Delete any preference files that were saved automatically.
3. Open this file “/Kit/gpgt/server/defaults.cs” in your favorite editor.
4. In this file locate the string “`$pref::Master[0]`”.

```
$pref::Master[0] = "2:master.garagegames.com:28002";
```

Now, you have two options. First, you may replace the “`$pref::Master[0]`” address, or second, you may add a new master server. For these exercises, I suggest adding a new master server.

At this time, please add a second master server address using the IP from your machine (we wrote this down above) . When you are done, your default.cs file should contain two lines similar to these, but using your IP address.

dumpMasterServersList()

Now, I'd like you to write another debug function. This function will, when executed, dump the values of the first 6 master server globals that it finds.

Using the following empty function body as a starting point, please add the code to loop between 0 and 5, printing the master server address for each index.

```
function dumpMasterServersList()
{
}
```

SERVER QUERIES

Testing Your Changes

Now, let's test our new changes and the new function. Please follow these steps.

1. Quit to the main menu.
2. Run the mission for this exercise and be sure “Host Multiplayer” is selected.
3. Open the console (~) and type the following.

```
dumpMasterServersList();
```

If you didn't see both master server addresses, please start this part of the exercise over and see if you missed a step. Once you're happy with the result, please move on and learn how to run the supplied master server.

SERVER QUERIES

4 Running a Master Server

As you learned in chapter 1, this guide supplies a master server (implemented in PERL, and written by Thomas Lund as a community resource). The file implementing this server can be found on your disk at “MasterServer/c3masterserver.pl”.

To run this script, you need to have PERL installed on your system. (You can download PERL for free from www.activestate.com.)

Assuming that you do have PERL installed, you can simply run the script and it will immediately start waiting for game servers and clients.

Changing Master Server Port

By default, this master server will list itself on port 28002. You can modify this by locating the following code in the file and changing the port to one of your preference.

```
my $PORT = 28002;
```

Test The Server

Please start the server at this time, either by double-clicking it or typing this on the command line (remember to cd to the proper folder/directory first).

```
perl c3masterserver.pl
```

If you are running Windows, you'll get a DOS window that looks like figure 4.2 on the next page.

SERVER QUERIES

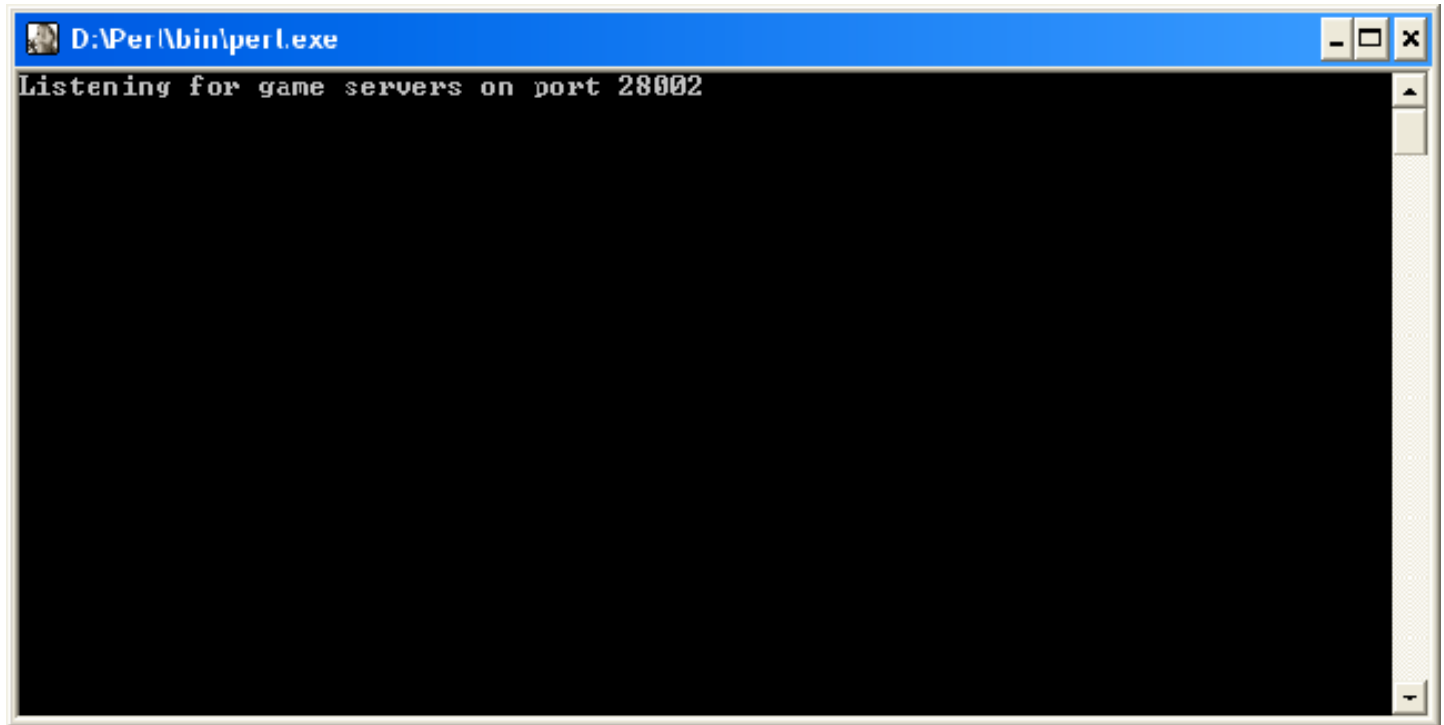


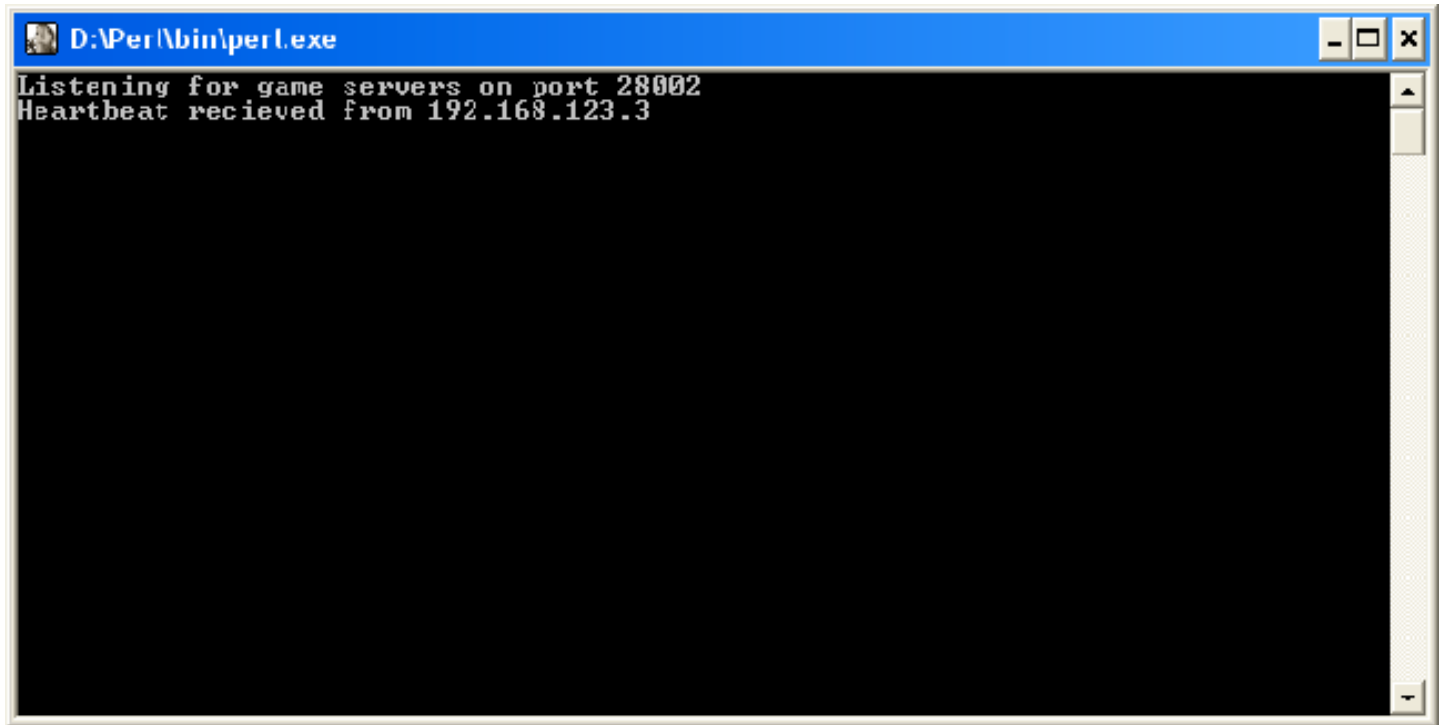
FIGURE 4.2 MASTER SERVER RUNNING IN WINDOWS XP.

As you can see, the server is sitting idle, just waiting. To see more results, please do the following.

1. Start the KIT.
2. Run the mission for this exercise and be sure that “Host Multiplayer” is checked.

As soon as the game starts up, you should see something like figure 4.3 on the next page.

SERVER QUERIES



```
D:\Per\bin\perl.exe
Listening for game servers on port 28002
Heartbeat recieved from 192.168.123.3
```

FIGURE 4.3 SERVER HEARTBEAT SENT TO MASTER SERVER.

SERVER QUERIES

5 Master Servers Query

In the last part of this exercise, we will write a function for querying the master server and try some experiments.

doMasterServersQuery()

This last query function is merely a wrapper for the console function `queryMasterServer()`. Your job in this is to take the skeleton I've provided and to fill it in with a call to `queryMasterServer()`, using the proper client query parameters (those we created early on in this exercise).

```
function doMasterServersQuery()  
{  
}
```

Testing Your Function

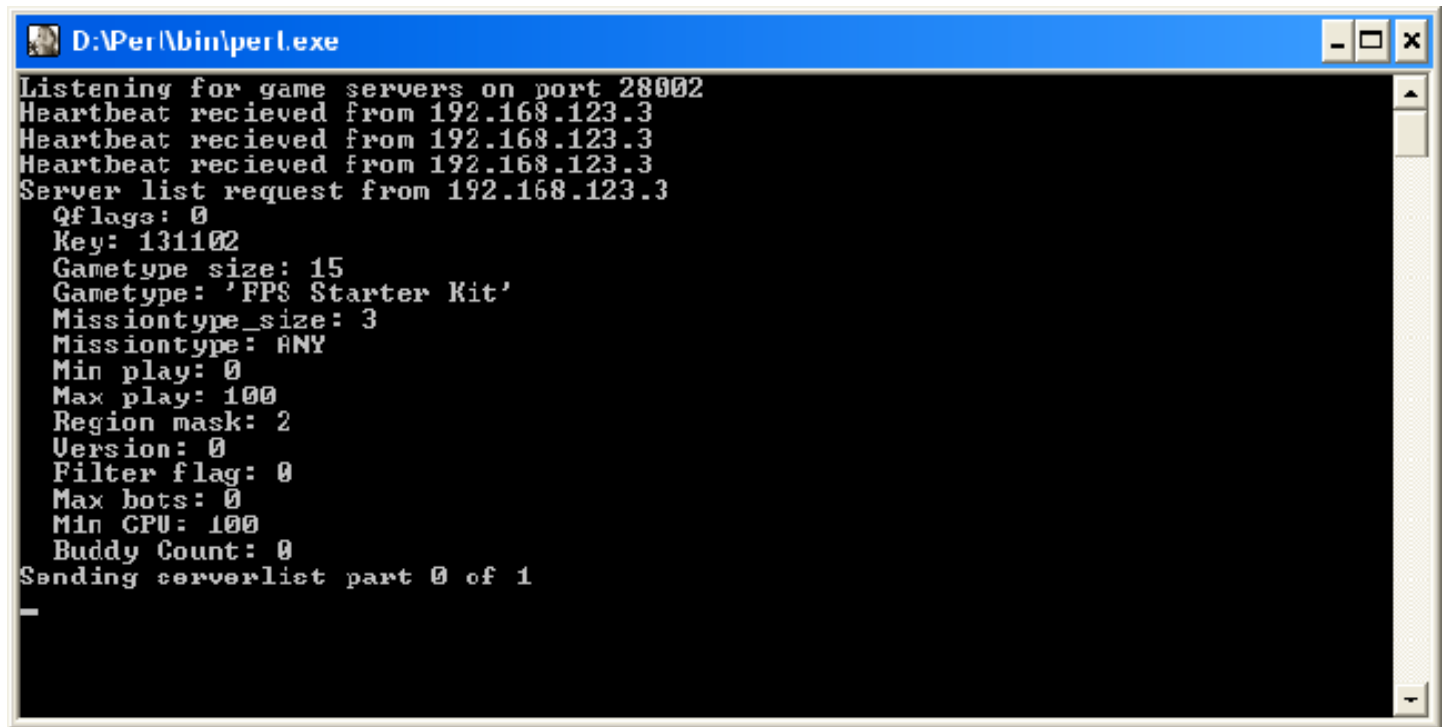
Now, let's test your new function. Please follow these steps.

- Exit the Kit.
- Start the Kit.
- Be sure the master server is running.
- Run exercise Chapter4 -> "001_Servers: Server Exercises" and select the "Host Multiplayer" checkbox.
- Open the console (~).
- Type the following:

```
doMasterServersQuery();  
dumpServerInfoData();
```

You should also see something like figure 4.4 in the master server console.

SERVER QUERIES



```
D:\Perl\bin\perl.exe
Listening for game servers on port 28002
Heartbeat recieved from 192.168.123.3
Heartbeat recieved from 192.168.123.3
Heartbeat recieved from 192.168.123.3
Server list request from 192.168.123.3
  Qflags: 0
  Key: 131102
  Gametype size: 15
  Gametype: 'FPS Starter Kit'
  Missiontype_size: 3
  Missiontype: ANY
  Min play: 0
  Max play: 100
  Region mask: 2
  Version: 0
  Filter flag: 0
  Max bots: 0
  Min CPU: 100
  Buddy Count: 0
Sending serverlist part 0 of 1
```

FIGURE 4.4 QUERY TO MASTER SERVER.

At this point, I suggest trying some experiments with the client query settings (globals) we created for this exercise. Specifically, try a query like this.

```
$Client::GameTypeQuery = "ANY";
$Client::MissionTypeQuery = "ANY";
doMasterServersQuery();
dumpServerInfoData();
```

Depending upon who is running servers at the time, you may get a huge list of servers from the Internet.