Programming Test

Along with this document is a binary executable (server.exe) which hosts the server for this test. It has been tested on Windows 7 & Windows 10 and requires a 64bit OS. The protocol for this server is detailed on the following page.

You will need to produce a UI in C++ or C# that will allow a user to issue commands to the server and see the responses. The look and feel of the UI is your choice, for example Console, WPF or MFC but should be clear and easy to understand.

You will have to submit source code and solution that can be compiled on Visual Studio 2012 or more recent; Visual Studio can be downloaded for free. We will be looking for clear coding style and OO principles and your code should be clearly commented where necessary.

You should also provide a document detailing any observations you may have on the behaviour of the server.

No support will be provided during the test.

Server protocol specification

The server will host on port 1234 on start-up and can be terminated by pressing ctrl-c

Some operations may take time to complete but this should not be allowed to degrade the user experience.

To issue a command to the server connect via TCP and send a byte that matches a valid command ID. The server will respond with a byte of 0 to indicate it is ready to process the command or an error. On success the command parameters can then be sent to the server.

There are 4 commands you can issue to the server.

Command ID	**Command**	**Details**
1	Reverse string	Send a string in a byte array terminated with a new line. The server will respond with the same string reversed.
2	Reverse integer	The server will reply with a uint32 which you will have to swap the higher and lower bytes and send back. The server will confirm success by sending 0 or failure by 0xFF.
3	Calculate pi	The server will expect a single byte to indicate how many decimal places to calculate pi and will respond with the value as a new line terminated string.
4	Add two integers	The server will expect two uint16 values. It will then sum them together and respond with a uint32 value which you should display.

In the event of a failure the follow error logic applies and should be handled gracefully to explain to the user what happened. The example bit pattern here shows an Invalid Argument at address 20.

Error Bit	Address 0	Address 1	Address 2	Address 3	Address 4	Code 0	Code 1
1	0	0	1	0	1	1	0

Error Bit is always set in the case of an error.

The address indicates the point of failure and will mean something to users of the system.

The Code maps to the following table

0: Out of Memory

1: Invalid Argument

2: Communication failure

3: Unsupported Command

Example exchange with server using command 4

Client Transmits	Server Transmits
Command Byte (0x4)	
	Command Response (0x0)
Data to process (4 5)	
	Data Response (9)