COEN 424 – Programming on Cloud (Fall 2017) Jafar Abbas (26346650)
Assignment 1 – Report October 26th

Introduction

In this assignment I wrote a client/server program to address the "price quote" scenario. The scenario is a client sends a 'Request for Quote' and the server replies a 'Response for Price'.

Requirements

1) Two (de)-serialization methods: Text-based and Binary.
2) Any software framework or standalone program for the server/client.
3) Any protocol for network communication.
4) RFQ & RFP definitions given
5) Any form of database

Specifications

1) C++ Programming language used
2) XML & Binary serialization using Cereal's C++11 open-source header-only serialization library
3) Standalone Windows console applications for client & server. One server process for binary and another XML serialization
4) UDP protocol was used for communication. Server-side two ports are listened to, one for binary and other for XML
5) A basic pre-arranged filesystem used to store information server-side

i) How to run my application

1) Extract the server and client into two different folders in a Windows environment

2) Open the server's 'Release' folder

3) Double-click 'coen424server.exe' under this will launch the first server (Binary listener)

4) Double-click 'coen424server.exe' again to launch the 2nd server (XML listener)

5) Open the client's 'Release' folder

6) Double-click 'coen424client.exe'

7) Following instructions to select desired serialization and enter your name, account-id and then select the RFQ option to enter the product category, product number and qty.

ii) Design of the data model

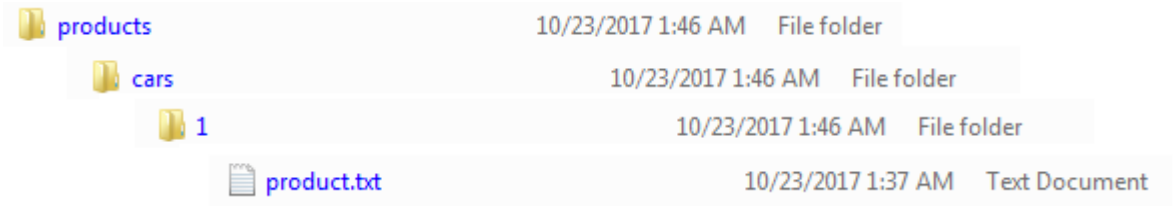
To represent my data during execution, I used a c++ structure for the RFQ and another for the RFP. Each struct contains the specified fields from the requirements.

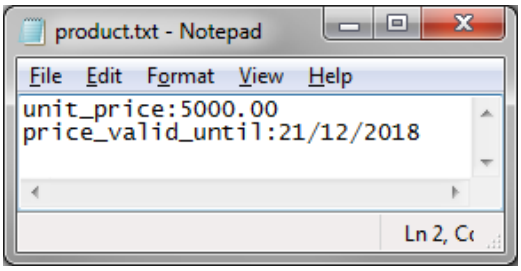
Then, I wrapped it with my own my_MSG structure. my_MSG contains a RFQ and a RFP. This is the structure that the program interacts with and serializes.

```
struct RFQ //request for quote
{
    int         account_id = -1;
    int         product_number = -1;
    std::string product_category = "";
    int         quantity = -1;

    template <class Archive>
    void serialize(Archive & ar)
    {
        ar(account_id, product_number, product_category, quantity);
    }
};

struct RFP //response for price
{
    float       unit_price = -1;
    std::string price_valid_period = "";

    template <class Archive>
    void serialize(Archive & ar)
    {
        ar(unit_price, price_valid_period);
    }
};

//structs to be serialized/deserialized
struct my_MSG
{
    std::string type = "";
    int         id = -1;
    int         port = -1;
    std::string addr = "";
    std::string name = "";
    std::string message = "";

    RFQ         request;
    RFP         response;

    template <class Archive>
    void serialize(Archive & ar)
    {
        ar(type, id, port, addr, name, message, request, response);
    }
};
```

As for storing my data server side, I used a simple file system architecture to hold records of my products, their unit price and the valid period.

Each category has its own folder. For each category, the different products are listed by number (product number) and with it is an associated text file that holds the unit price and valid period as seen below:





iii) Methods used for data serialization / de-serialization

For text-based data serialization, I used XML.

For binary data serialization, I used 'Portable Binary'.

iv) How data model design and serialization methods are applied in the data communication

my_MSG contains a RFQ and RFP and some extra information. This structure is understood and serializable/deserializable by both the server and the client.

The idea is my_MSG contains information such as "type", "message", "ID" so that the client and server can have a more sophisticated communication protocol built on top of UDP. For example, types can be "RFQ", "RFP" but it could also be "ERROR" in the case that the RFQ contains invalid information. The message field can store a more detailed error message.

my_MSG would also have as a payload a RFQ and RFP since they're fairly light-weight structures.

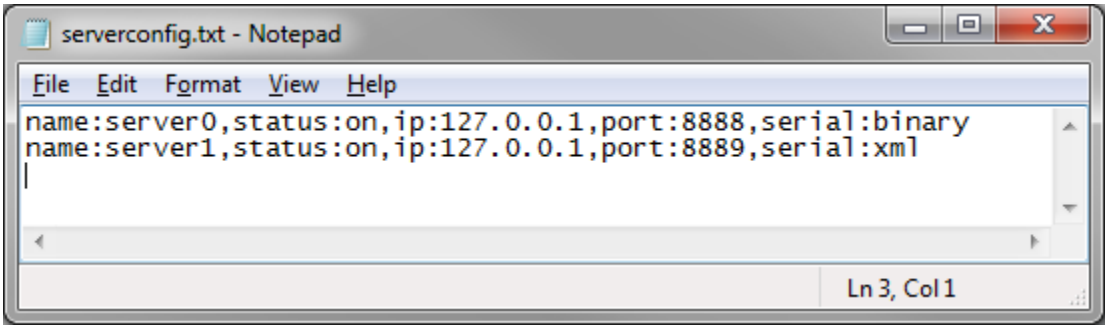
v) Libraries used for data serialization

I used cereal (https://uscilab.github.io/cereal/), which is a light-weight, header-only, easy to use C++11 library for serialization. It doesn't require special data-model preparation.

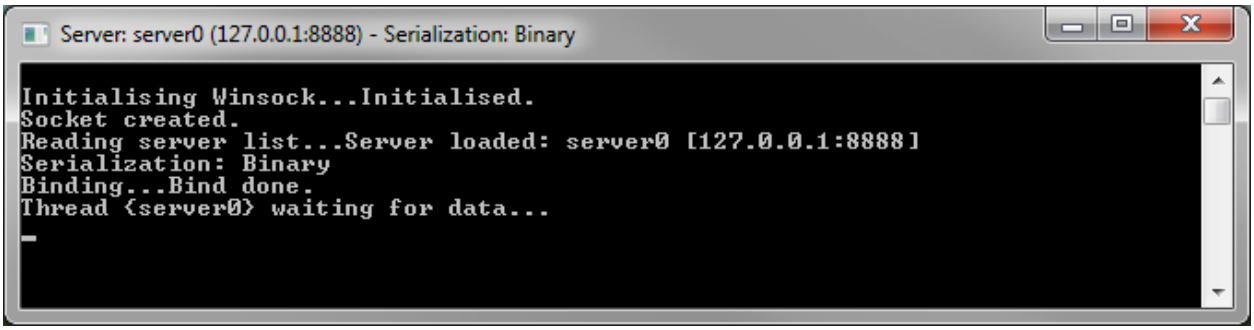
It supports JSON, XML, Binary, Portable Binary and it also supports c++ file streams or string streams. The data model does not need to be changed for the different serialization methods. My experience was very positive with the library. The only con is its limited binary serialization methods.

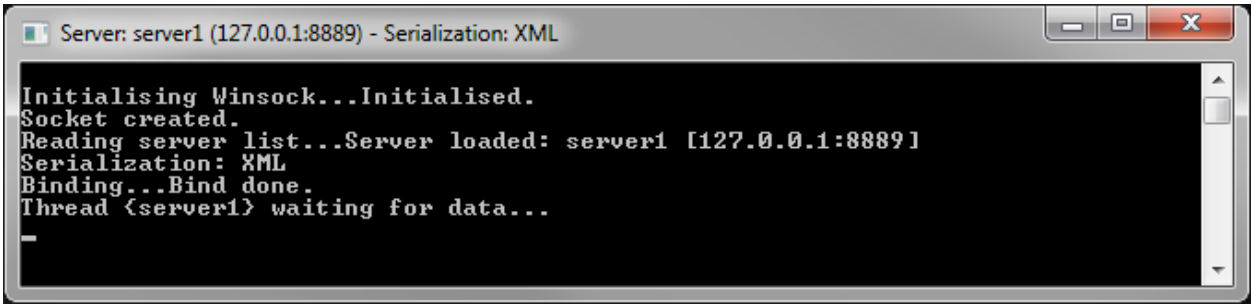
vi) Screenshots of running the application

Server side config file:

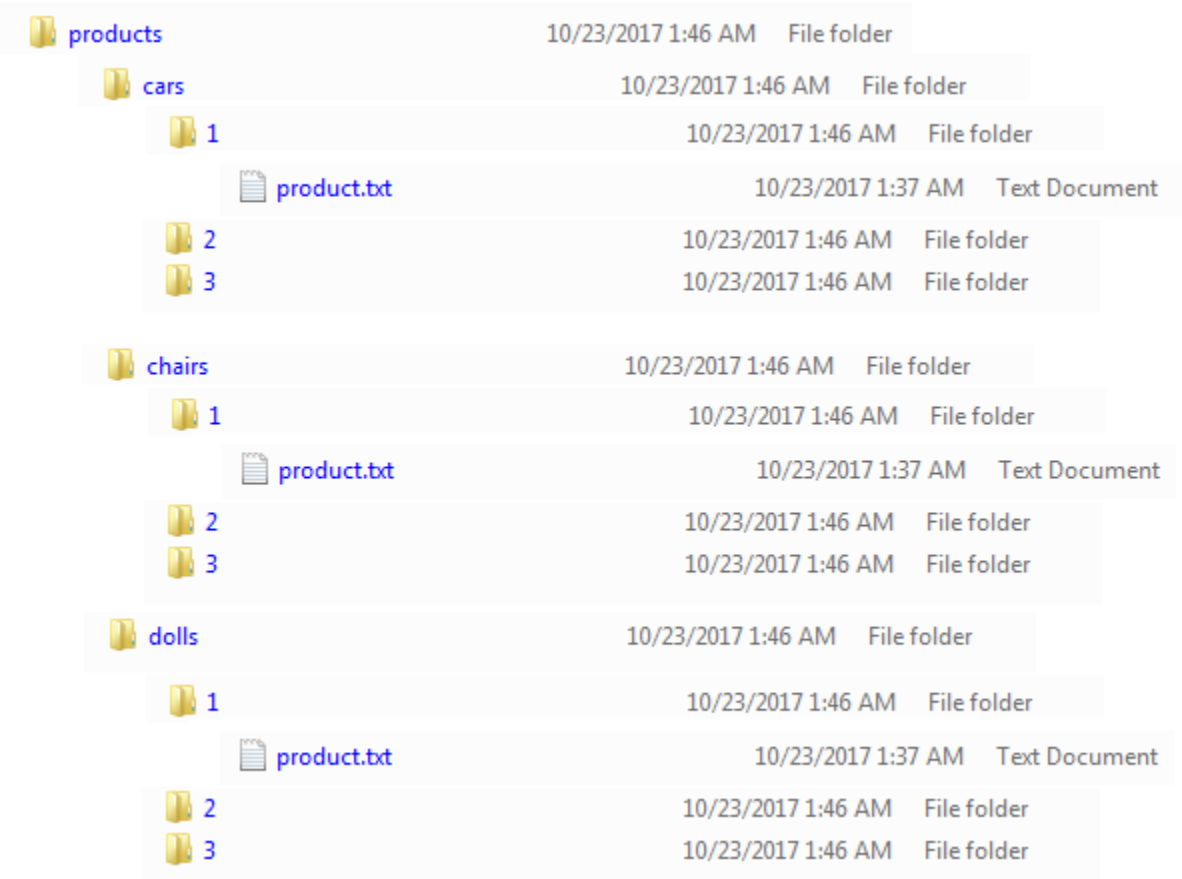


Initializing servers for binary (port 8888) and XML (port 8889)

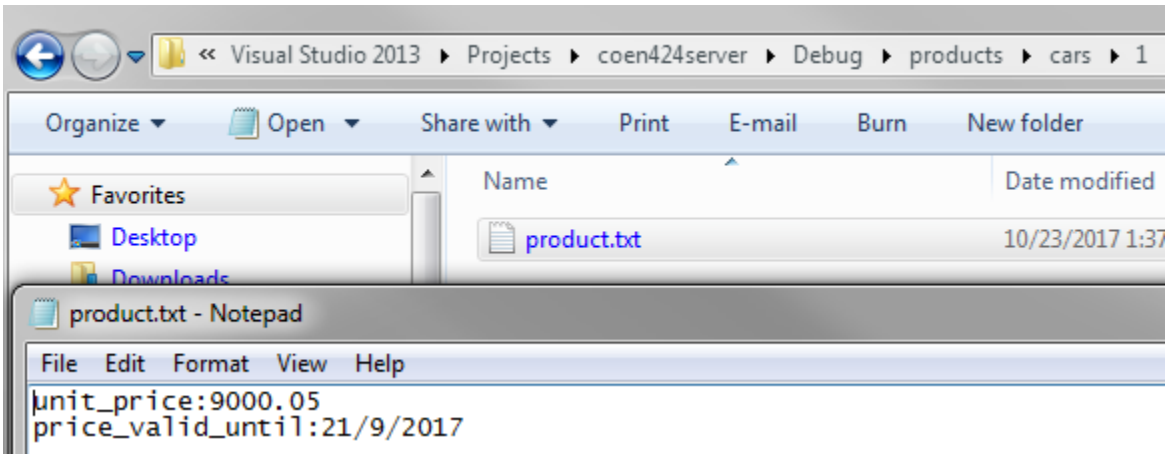




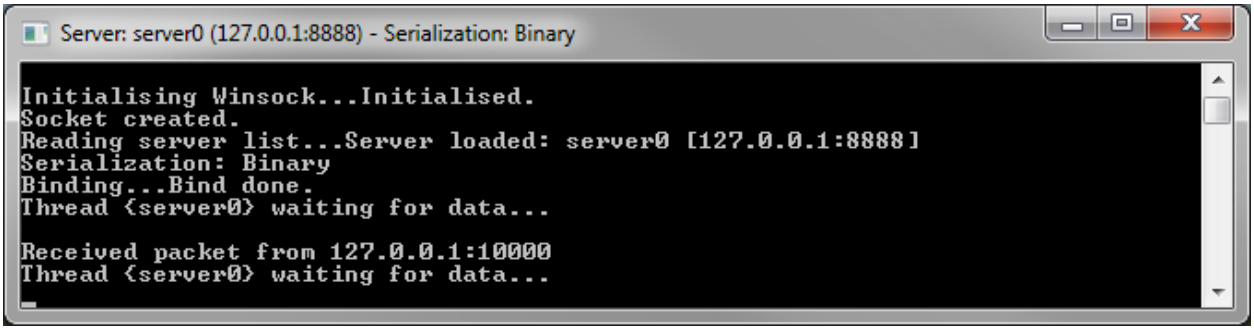
Storage organization based on category and product number:

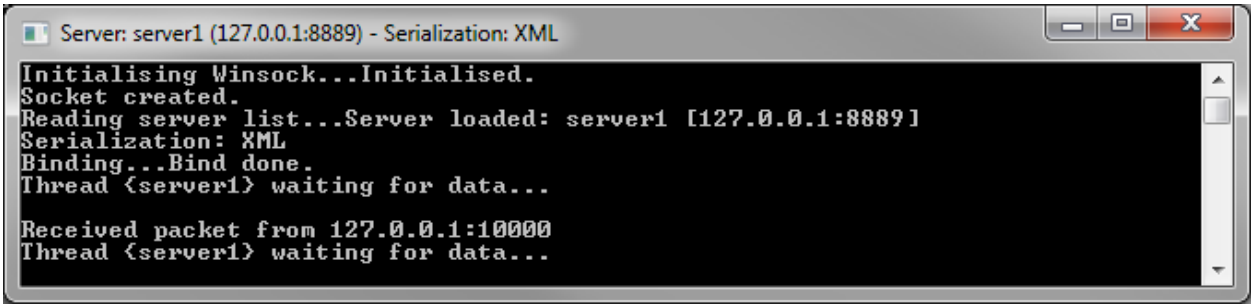


product.txt holds the unit price and valid period:

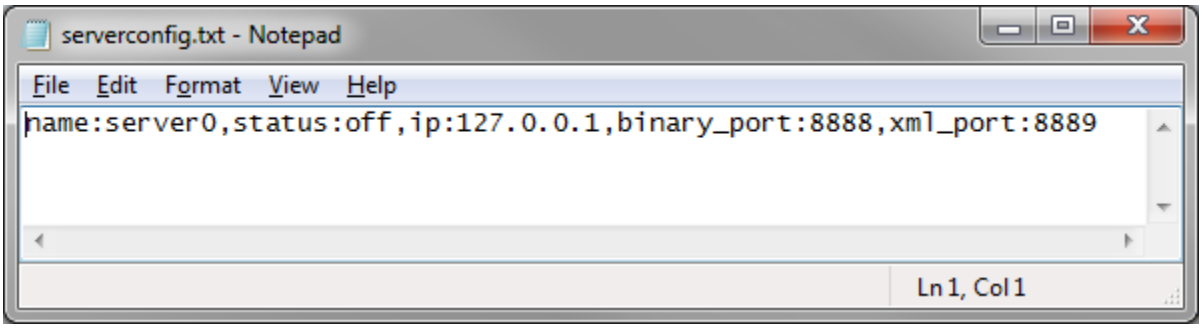


Server-side: **Listener threads receiving packet**

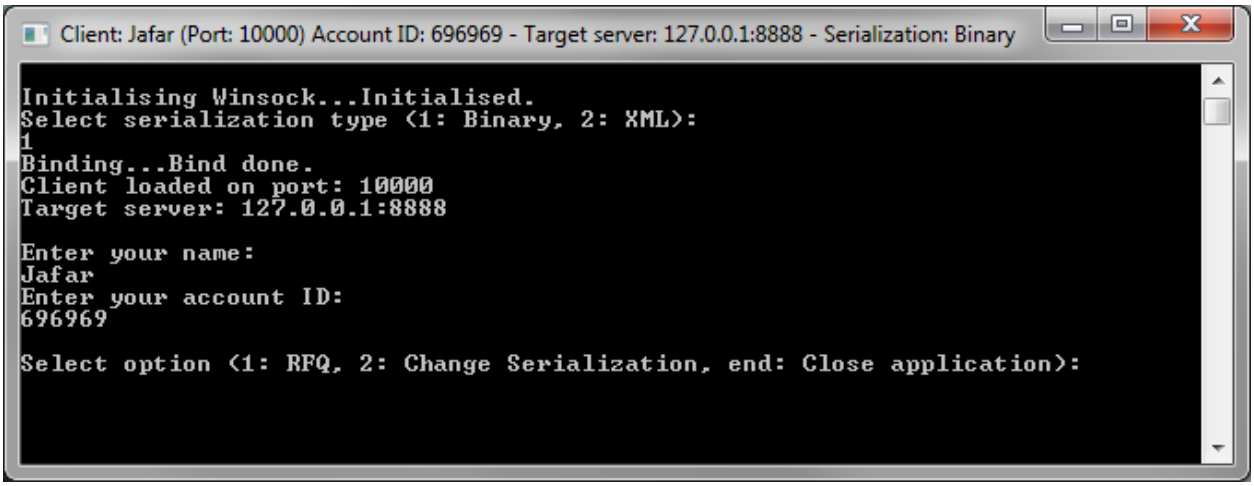




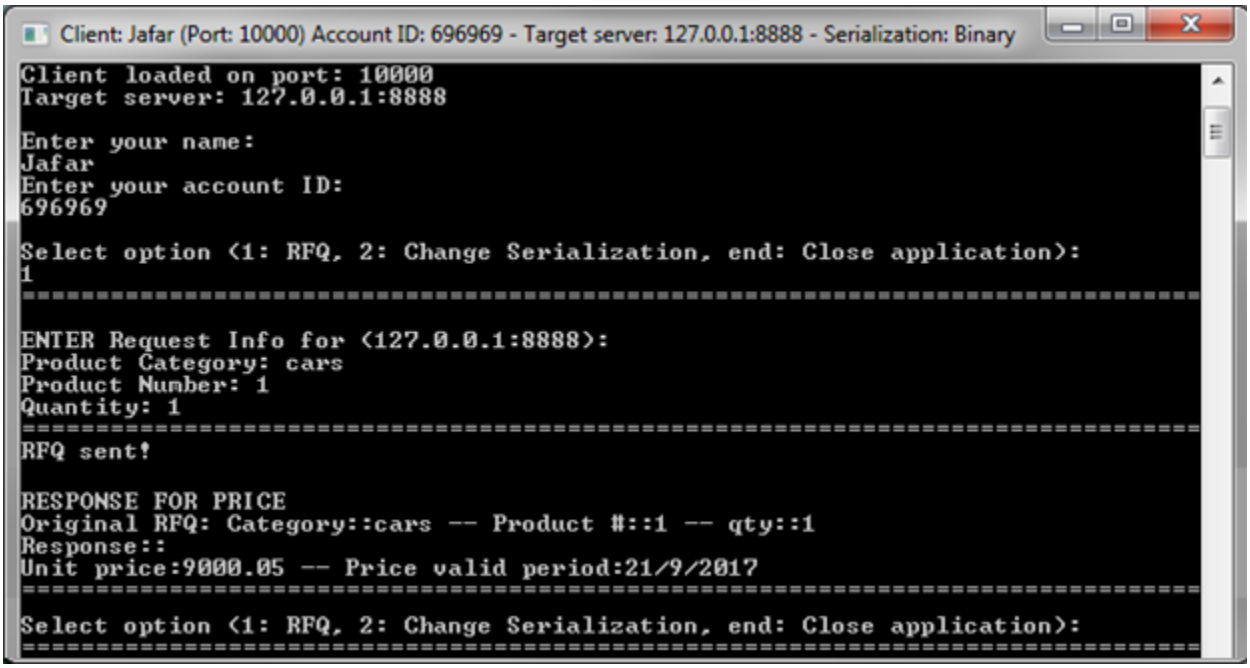
Client side configuration file, which holds the address and port to reach the servers:



Client-side initialization, choosing serialization type and entering your name, account ID.



Filling out a request for quote form and **receiving a response** (using Binary):



Filling out a request for quote form and **receiving a response** (using XML):

```
Client: Jafar (Port: 10000) Account ID: 696969 - Target server: 127.0.0.1:8889 - Serialization: XML
Response::
Unit price:9000.05 -- Price valid period:21/9/2017
==============================================================================

2
Select serialization type (1: Binary, 2: XML):
2

Select option (1: RFQ, 2: Change Serialization, end: Close application):
1
==============================================================================

ENTER Request Info for (127.0.0.1:8889):
Product Category: cars
Product Number: 1
Quantity: 1
==============================================================================
RFQ sent!

RESPONSE FOR PRICE
Original RFQ: Category::cars -- Product #::1 -- qty::1
Response::
Unit price:9000.05 -- Price valid period:21/9/2017
==============================================================================


Select option (1: RFQ, 2: Change Serialization, end: Close application):
==============================================================================
```

Viewing XML Serialized Data:

XML: Server side

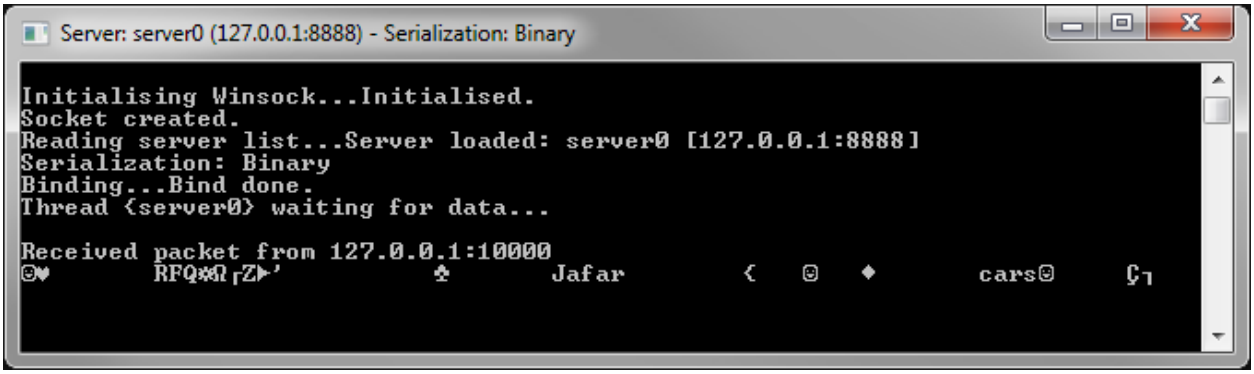
```
Server: server1 (127.0.0.1:8889) - Serialization: XML

Initialising Winsock...Initialised.
Socket created.
Reading server list...Server loaded: server1 [127.0.0.1:8889]
Serialization: XML
Binding...Bind done.
Thread <server1> waiting for data...

Received packet from 127.0.0.1:10000
<?xml version="1.0" encoding="utf-8"?>
<cereal>
        <value0>
                <value0>RFQ</value0>
                <value1>1523549042</value1>
                <value2>10000</value2>
                <value3></value3>
                <value4>Jafar</value4>
                <value5></value5>
                <value6>
                        <value0>12345</value0>
                        <value1>1</value1>
                        <value2>cars</value2>
                        <value3>1</value3>
                </value6>
                <value7>
                        <value0>-1</value0>
                        <value1></value1>
                </value7>
        </value0>
</cereal>

Thread <server1> waiting for data...
```

XML: Client side

```
Client: Jafar (Port: 10000) Account ID: 12345 - Target server: 127.0.0.1:8889 - Serialization: XML

1
==============================================================================

ENTER Request Info for <127.0.0.1:8889>:
Product Category: cars
Product Number: 1
Quantity: 1
==============================================================================
RFQ sent!

Select option (1: RFQ, 2: Change Serialization, end: Close application):
<?xml version="1.0" encoding="utf-8"?>
<cereal>
        <value0>
                <value0>RFP</value0>
                <value1>1523549042</value1>
                <value2>8889</value2>
                <value3>127.0.0.1</value3>
                <value4>Jafar</value4>
                <value5></value5>
                <value6>
                        <value0>12345</value0>
                        <value1>1</value1>
                        <value2>cars</value2>
                        <value3>1</value3>
                </value6>
                <value7>
                        <value0>9000.0498046875</value0>
                        <value1>21/9/2017</value1>
                </value7>
        </value0>
</cereal>

==============================================================================

RESPONSE FOR PRICE
Original RFQ: Category::cars -- Product #::1 -- qty::1
Response::
Unit price:9000.05 -- Price valid period:21/9/2017
==============================================================================
```

Viewing Binary Serialized data:

Binary: Server side:



Binary: Client side:

Error handling:

Wrong Category or product number:

```
==============================================================================

ENTER Request Info for (127.0.0.1:8888):
Product Category: fake_cat
Product Number: 1
Quantity: 1
==============================================================================
RFQ sent!

ERROR::PRODUCT NOT FOUND
Original RFQ: Category::fake_cat -- Product #::1 -- qty::1
```

Server not online (client re-sends un-answered RFQ messages):

```
recvfrom() failed with error code : 10054

Resending (1) following timed out messages:
type::RFQ
id::1524754879
port::8888
addr::127.0.0.1
name::jafar
acc_id::123 -- product_category::test -- product_#::1 -- qty::3
recvfrom() failed with error code : 10054
```