

## DataPump Example written in C#

Copyright Stewart Button

13th August 2010

This software sends random historical bars to Neuroshell Trader (NST), along with ticks and future bars. It closes when the NST closes and can also retrieve the user's username and password.

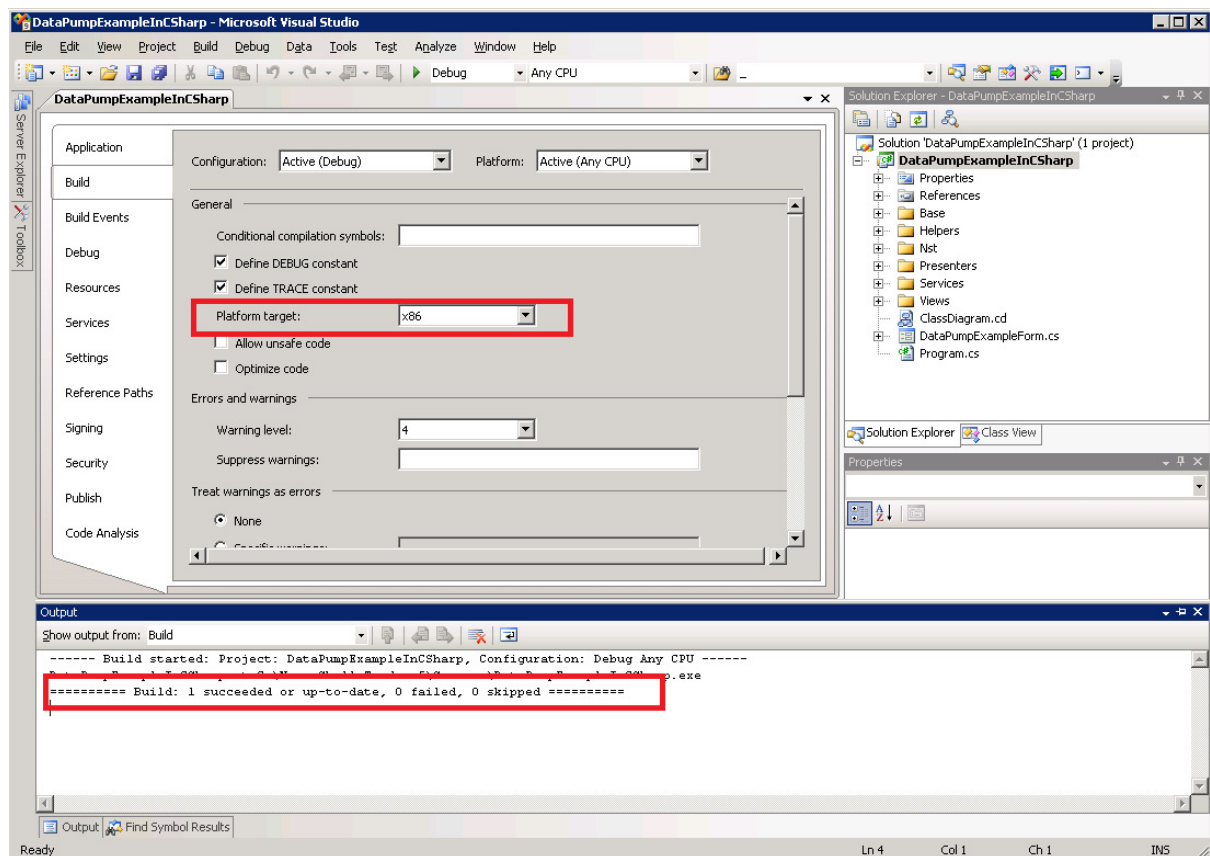
This solution provides a working example of a NST DataPump written in C# which interacts via the Ward's NSTFeed.dll API. The Model-View-Presenter (MVP) pattern has been chosen to achieve clarity, however this is by no means a necessity for your solution.

The best way to understand this example is to build the solution, deploy it to NST, use it and finally examine the code once you understand what it does. The notes below will give you a feel for how to build and install it.

### Installing and Usage

The following steps should give you a good feel for how the solution works.

- 1) Firstly we start by compiling this solution so that we can see it running. Go ahead and compile this solution to create a .exe. One important thing you must do is to set the compile option to compile targeting a 32bit environment.

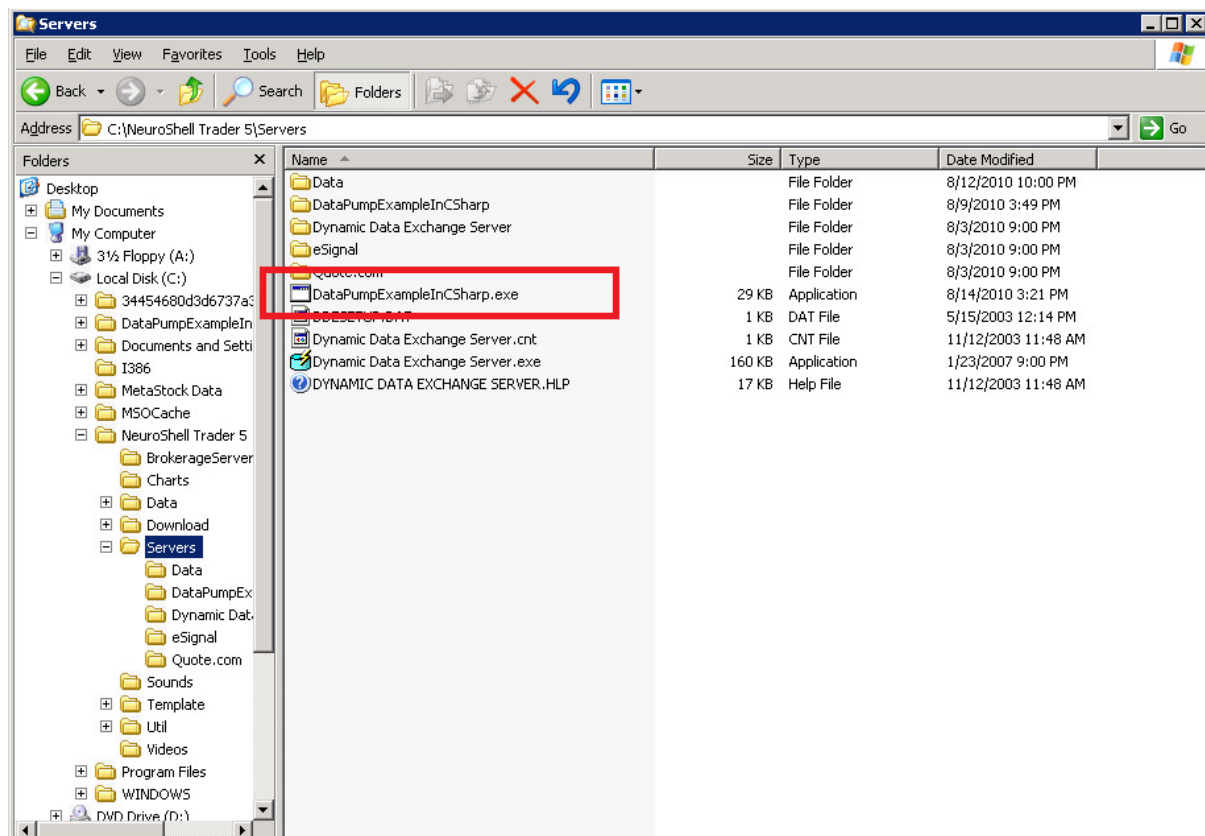


That is set:

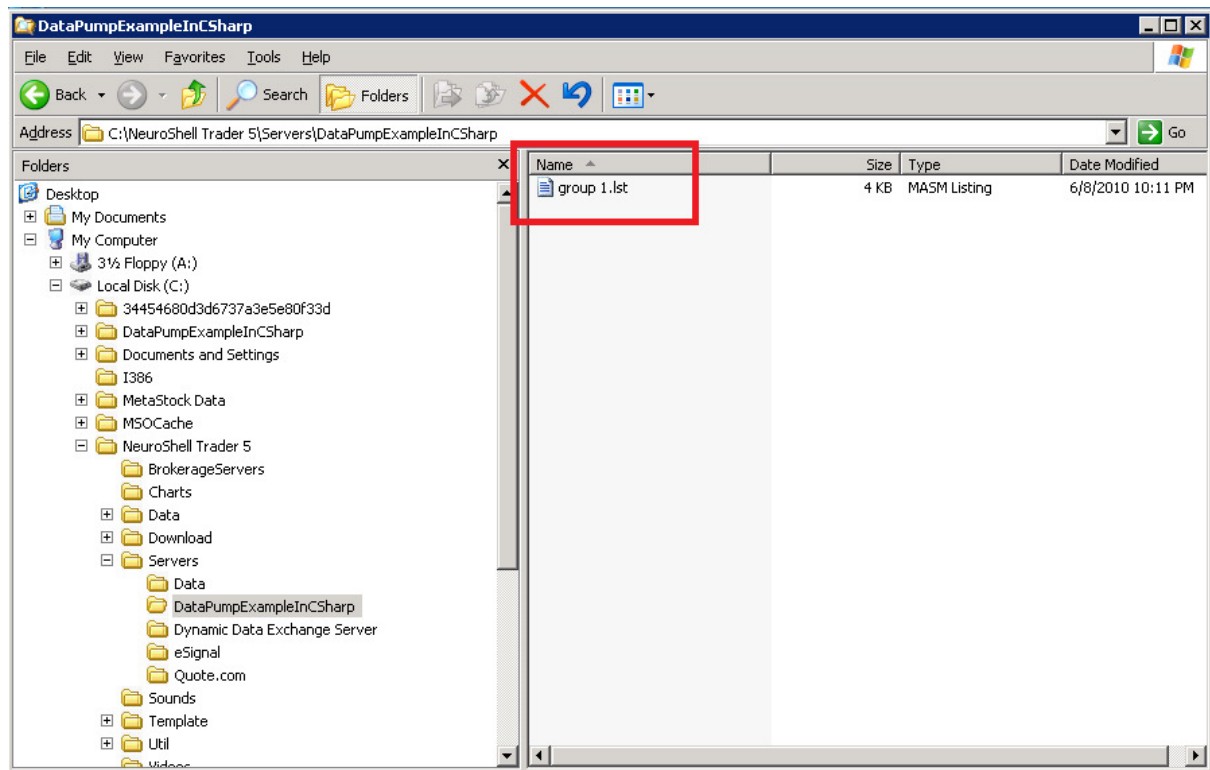
Project -> Properties -> Build -> Platform target = x86

If you don't do this then your solution will expect to find a 64bit version of the NSTFeed.dll file which of course doesn't exist. So, it might sound counter-intuitive but you need to set it to run in 32bit mode for it to run on a 64bit computer.

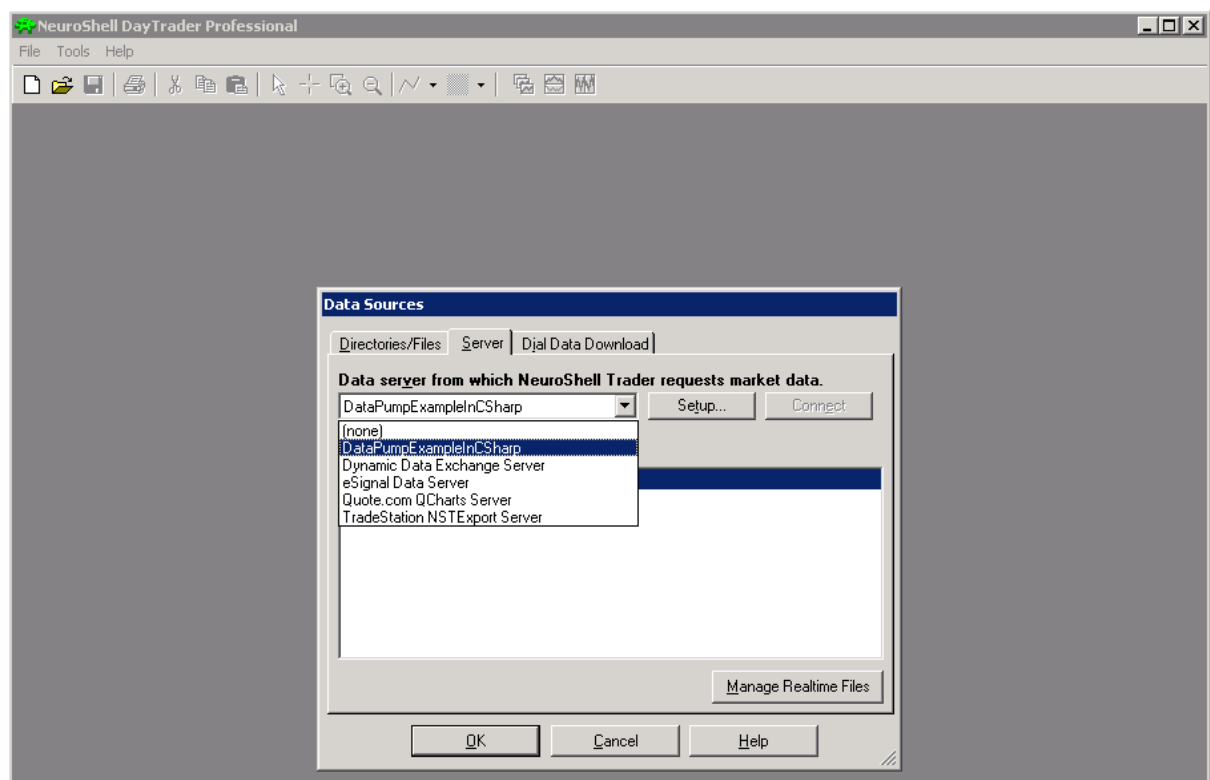
2) Once you have compiled your solution, copy the DataPumpExampleInCSharp.exe from your bin directory to your C:/Neuroshell Trader/Server/ directory.



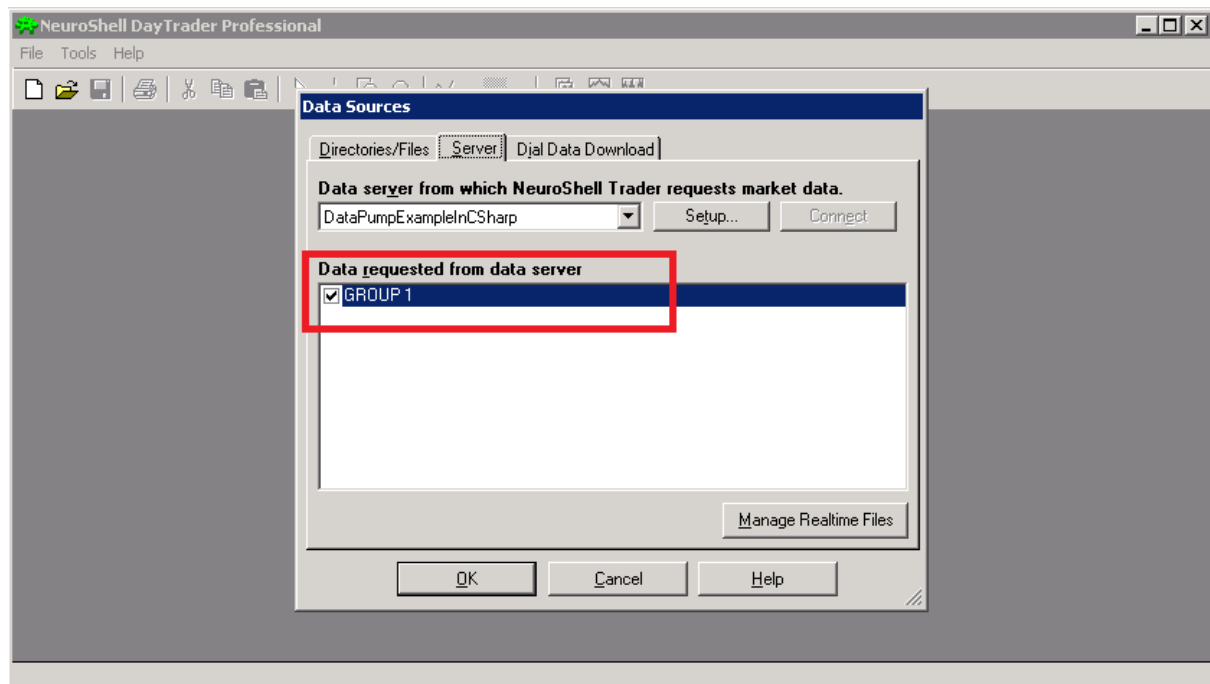
Create a new directory C:/Neuroshell Trader/Server/DataPumpExampleInCSharp. This must match the name of the exe. Copy the group 1.lst file to this new directory. This is now ready to use.



3) Launch NST and go to Tools -> Data Sources and choose DataPumpExampleInCSharp as your DataSource.



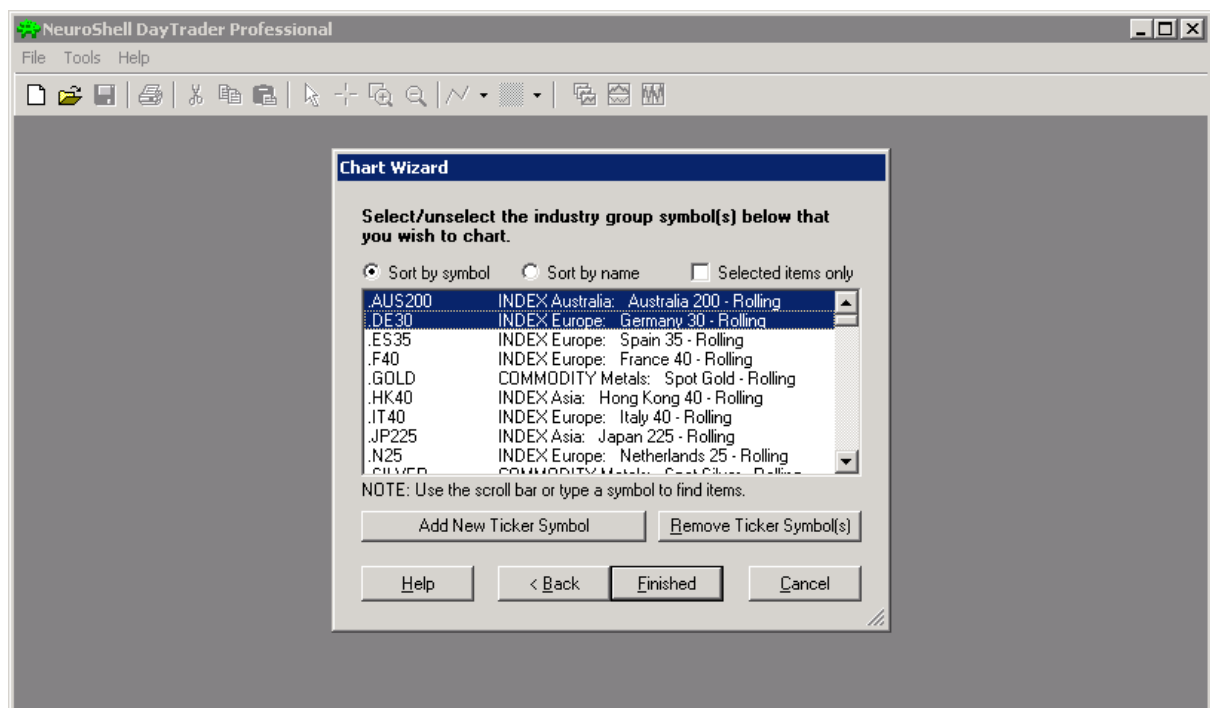
You should see a checkbox for the GROUP 1 list file. Check this to confirm that you want those securities.



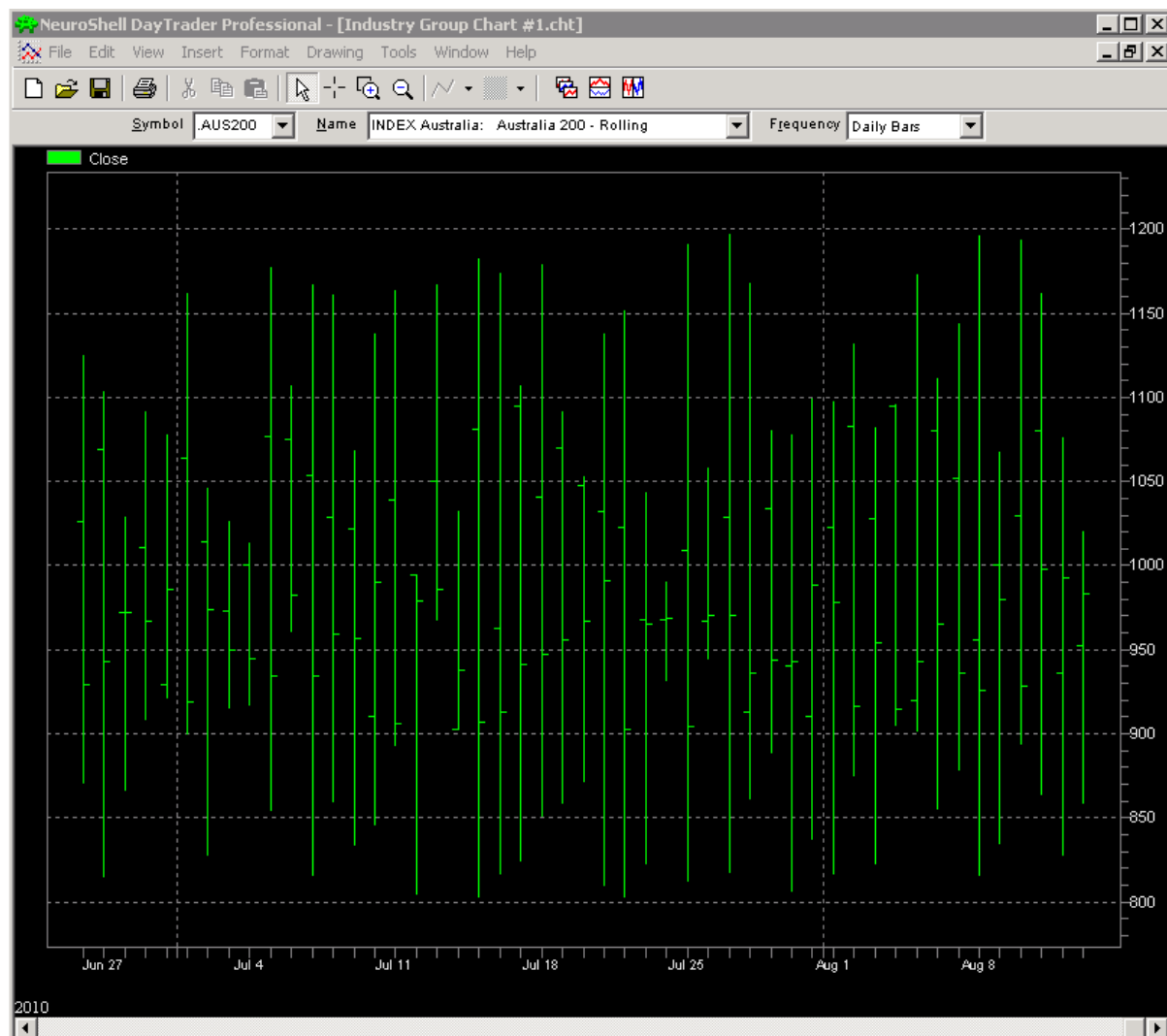
Click Setup and choose a username, password and set the timezone to Eastern Time.

IMPORTANT: Click OK and then close NST completely. NST will then be configured to use this DataPump.

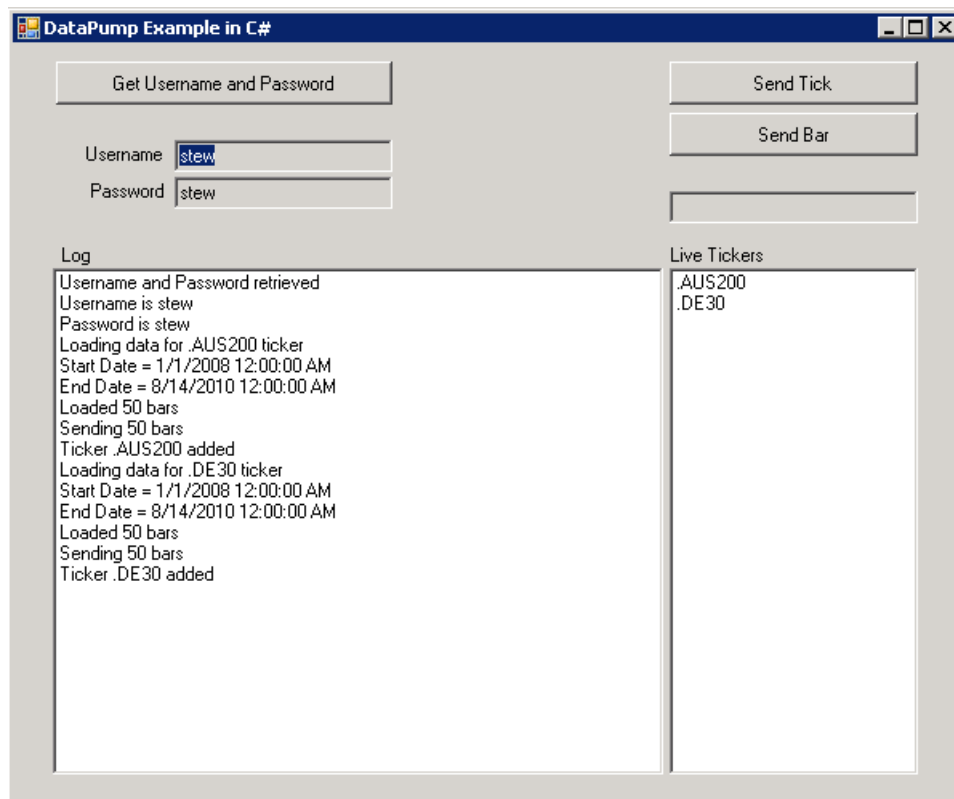
4) Open NST and create a new Daily chart and choose the .AUS200 and .DE30 indices from the Industry Group category (a misnomer I agree but Industry Groups is a category which you are unlikely to already have instruments in).



Once you have clicked Finish then give NST about 15-30 seconds thinking time. This is usual for the first request. Subsequent sending of data should be much faster. You can see that random bars have been loaded for these two indices.

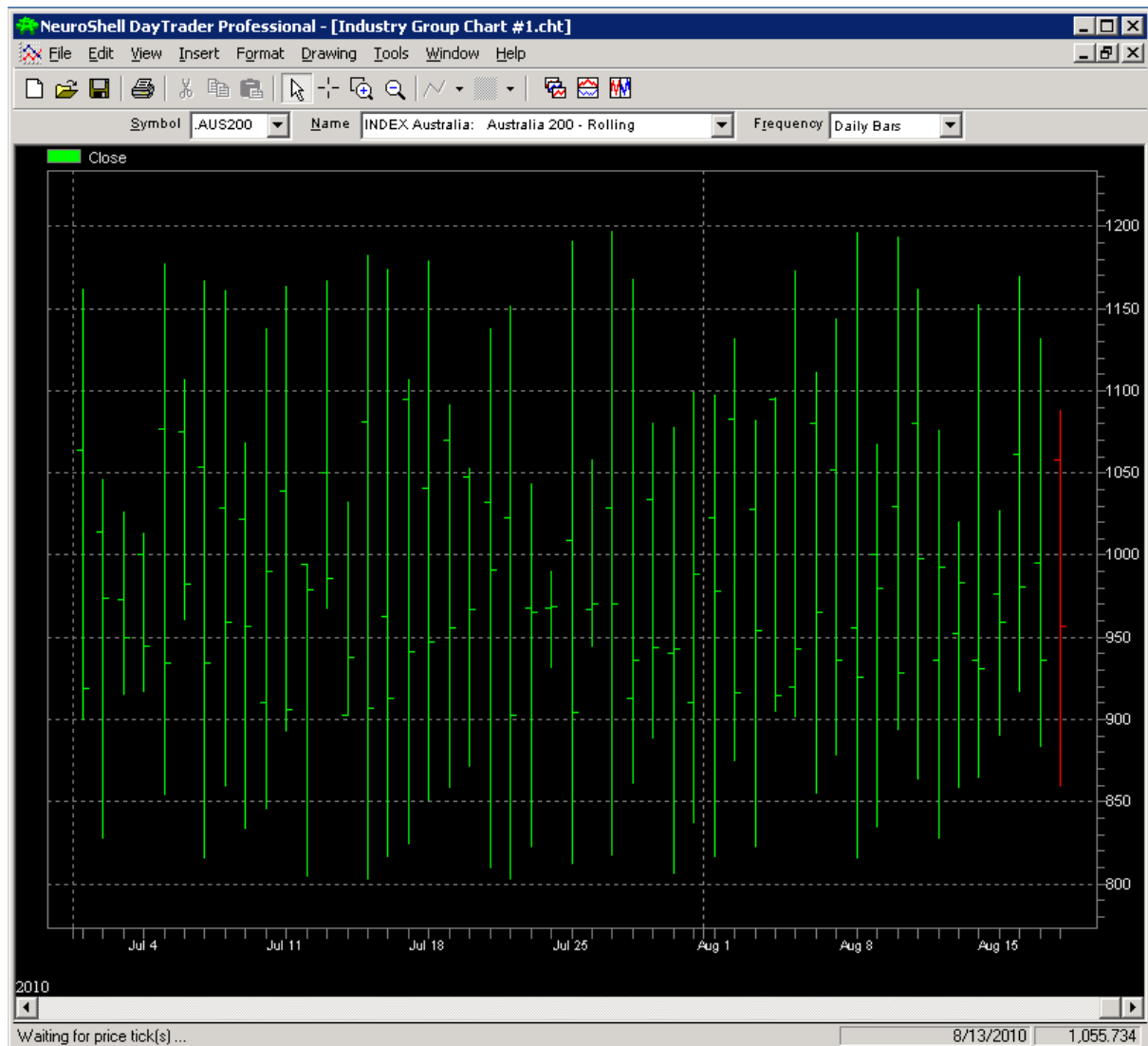


You will also be able to see the DataPump program. Note the Log of communication on the left and on the right the two tickers that are loaded.



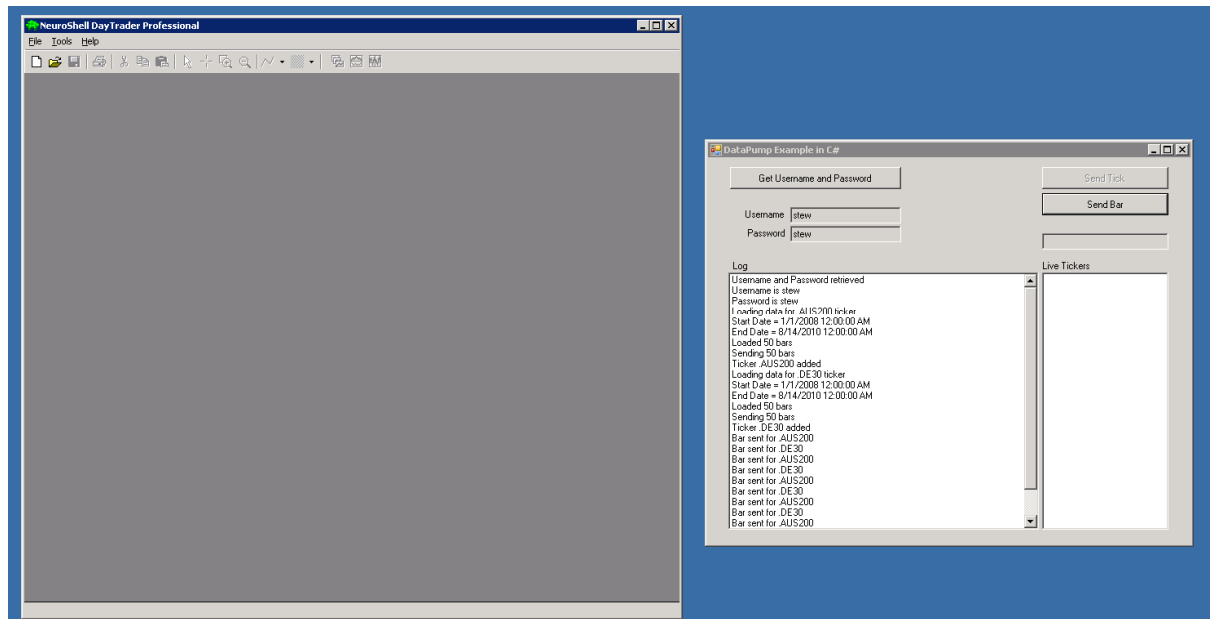
5) Go ahead and click the Get Username and Password button. NST will respond by updating the Username and Password fields if they are not already set.

You can choose to send future bars or ticks by using the buttons in the top right. Try one or the other. Note that you will have to shut NST and re-launch it in order to use the other. It doesn't make sense to mix bars and ticks (or at least it is more difficult to program in this example).



Note that the current bar is shown in red.

6) Close the NST chart. You will see that the two tickers will be removed from the DataPump program.



7) Finally, close NST and you will see that the DataPump program also closes.



## Key Reusable Components

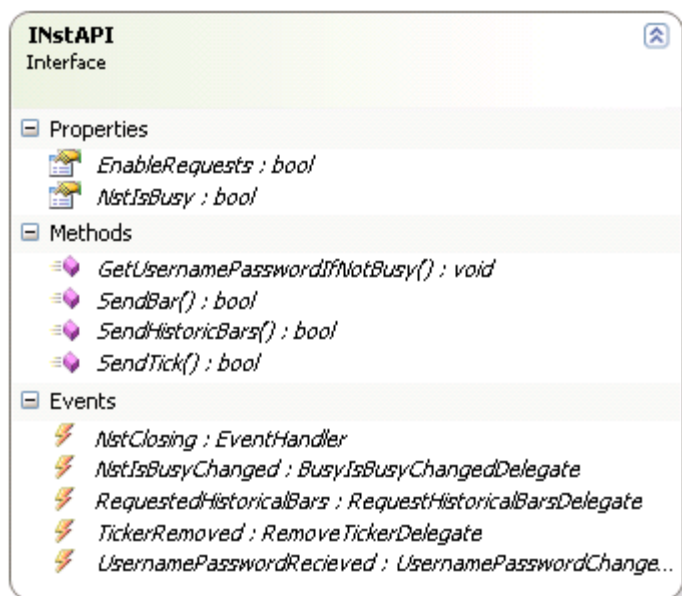
### *NstFeed.cs*

In the NST folder/namespace you will find NstFeed.cs. This allows the DataPump program to communicate via your Windows operating system with NST. You will need this file however need not need not concern yourself with the details.

### *NstAPI.cs*

This file is the wrapper around calling NST. Once again, you will require this file and need not be too concerned with the programming other than to realise that communication to NST is made by calling NstFeed and communication from NST is made via events published to a TextBox which we handle the Changed event on.

### *INstAPI.cs*



This interface describes the public members of NstAPI. This is what you will program against.

Start by instantiating a reference to NST and attach handlers to deal with NST events.

```
void View_Load(object sender, EventArgs e)
{
    // Instantiate the NstAPI(require username and password)
    _nst = new NstAPI(true);
    _nst.RequestedHistoricalBars += Nst_RequestedHistoricalBars;
    _nst.UsernamePasswordRecieved += Nst_UsernamePasswordSet;
    _nst.NstIsBusyChanged += Nst_BusyChanged;
    _nst.TickerRemoved += Nst_TickerRemoved;
    _nst.NstClosing += Nst_NstClosing;
    _nst.EnableRequests = true;
}
```

Below are the Events, Properties and Methods that you will use to communicate with NST. These are (arguably) much cleaner, more intuitive and easier to manage than those in the original NstFeed.dll interface.

Event	RequestedHistoricalBars	NST is asking for historic data. You will also want to keep note of the ticker and timeframe so that you can also supply new live data.
Event	TickerRemoved	NST is notifying that it is not interested in this ticker anymore as the user has closed the chart.
Event	UsernamePasswordRecieved	NST is notifying what the current username and password are. This is usually triggered by a call to the GetUsernamePasswordIfNotBusy method.
Event	NstIsBusyChanged	NST is notifying that it is busy. During this time it is best not to send tickers etc. For example, the user opens a chart and while the broker is sending historical data NstIsBusyChanged will be raised to notify that it is busy.
Event	NstClosing	NST is notifying that it is closing. You will want to close your DataPump at this point.
Property	EnableRequests	This property enables NST to communicate with our DataPump. Generally you will set this to true.
Property	NstIsBusy	Indicates if NST is busy. It's best not to call methods while NST is busy.
Method	GetUsernamePasswordIfNotBusy	Tells NST to raise the UsernamePasswordRecieved event.
Method	SendHistoricBars	Sends historic bars to NST. Contrary to the DataPump documentation, this should only be done once NST has asked for them.
Method	SendTick	Once historic bars have been sent to NST you may wish to send a tick.
Method	SendBar	Once historic bars have been sent to NST you may wish to send a full bar.

### Assumptions/Known Issues

- This demo may not work for multiple charts open at once which have different time periods. The NstAPI is capable of this; however I have chosen to keep the example simple.
- This demo may not work for Monthly charts.
- I generally use a fixed set of timeframes, eg 1min, 5min, 15min, 1 hour etc. I'd recommend doing the same and putting these into the BarType enum rather than making your DataPump work for any timeframe.
- Intentionally, a maximum of 50 bars of random history are produced regardless of how long a timespan is chosen in NST. This is for performance reasons and to keep the example simple. The NstAPI is however capable of any number (within reason) though.
- NST will often take 15-30 seconds or so to digest the first set of data sent to it. This is normal.
- As a separate program of mine which is not included, I have taken NstAPI contained in this example and built my own interface to my broker (GFT Global Markets). The DataPump program described in this document is only really useful to you as a quick start example. However if you take NstAPI from it and program against it, you too can build a C# DataPump to communicate with your broker. Unfortunately GFT do not allow the details of their API to be published here.

---

*Stewart Button is not responsible for any losses occurring from ideas communicated in this document. This is not meant to represent financial advice in any sense.*