

Algorithmica Research AB

Using the Inter-Quantlab Communication Server (IQC) using Quantlab 3.0

Using the Inter-Quantlab Communication Server (IQC)

Some applications have the need of sharing information between them. There are many ways of solving such user interaction depending on the available it-infrastructure. A common method is by using a common database where users can read-and-write information. For some applications where data is of a streaming type with very frequent updates a more direct communication might be better.

Having an IQC server in place will create such a direct communication bridge between users of the Quantlab clients, regardless if they are using Quantlab through an Excel sheet or direct.

The IQC server will mimic a real-time source feed such as Reuters or Bloomberg. The difference is, of course, that you have to provide the IQC with your own streaming data coming from a Quantlab user within the community.

To get some feel for what the IQC can do we will look at two different examples. First we will create a chat room where Quantlab users can send and receive messages to and from a bulletin board. Secondly we will create a market data feed where a market maker can internally distribute some spreads from an illiquid bond pricer.

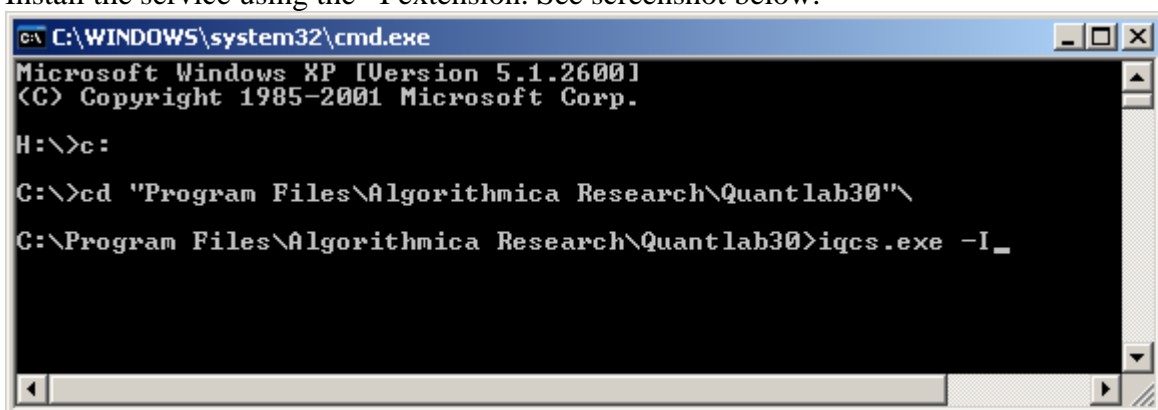
First we will look at how to install the IQC server

Step-by-step installation of the IQC on the server

The IQC server is only needed on one server/pc. All Quantlab clients can then communicate using the same server node.

Using the command line - go the folder containing the iqcs.exe programme.

Install the service using the -I extension. See screenshot below.

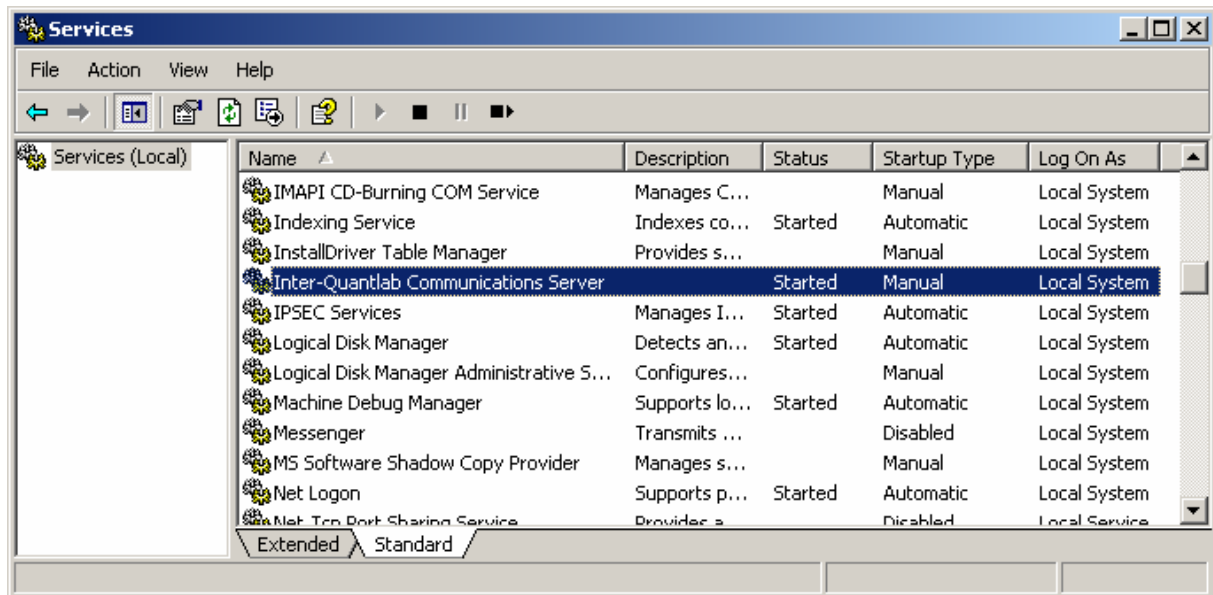


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>c:

C:\>cd "Program Files\Algorithmica Research\Quantlab30\"
C:\Program Files\Algorithmica Research\Quantlab30>iqcs.exe -I_
```

Go to the services window in the control panel and start the service.



Creating a connection to the IQC server from the Quantlab client

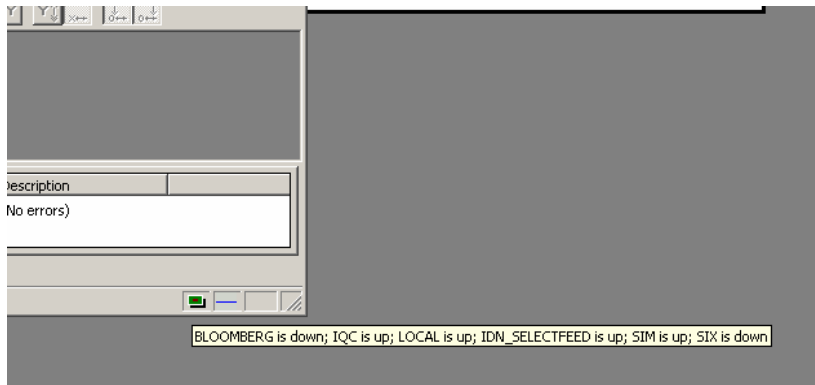
In the same folder as the Quantlab.exe there should be a file called iqc24.qrt or iqc30.qrt depending on the version of Quantlab. This file is the local communication program that will give the user/programmer the function library used for reading and publishing information to the central IQC node.

In the qlab30.ini file the following tag will tell Quantlab that there is an additional real-time source available. In this example the IQC service was installed on a server called "qlbhill". If the installation of the IQC service was on your local pc, this would be the name of your pc.

```
iqc {
    dll = 'iqc30.qrt'
    feed = 'IQC'
    server = 'qlbhill'
}
```

For Quantlab 2.4x the equivalent information is found in the registry in the rt tag.

Now we are ready to start Quantlab and see in the lower right hand corner (green icon) that the IQC is connected as a real-time source.



Creating a chat room using IQC

Let's start with writing some code to publish rows to the chat.

```
out string send(string user, out string message)
{
    string result = message;

    if (!null(message) || message != "") {
        iqc_publish("IQC", "chat", [ "user", "time", "message" ],
            [ user, sub_string(str(now()), 11, 8), message ]);

        message = "";
    }

    return result;
}
```

We create a function that takes the name of the user and a message as input. The message we declare as an “out” parameter which means that it will be called by reference. In the user interface we can then clear the message box as we reference the message variable and set it to an empty string.

The `iqc_publish` function takes four input arguments;

- 1) the name of the feed (here “IQC”)
- 2) the name of the `iqc` identifier that will hold our information (here “chat”)
- 3) a vector or field identifiers for the different bits of information in the `iqc` identifier
- 4) a corresponding vector with data for each field in the identifier.

As information we send three strings to the “chat” `iqc`-identifier each time the function is called. This information will replace the old information that was last updated in the same way as the last price of a stock coming in the market data feed.

After we have published the user name, timestamp, and message we clear the message.

In order to keep track of the history of the chat we can now create a function that will subscribe to the `iqc` identifier and its fields and then store all incoming messages in a local vector.

We do not need to ask the iqc server if there is any new information. The iqc server will push any new messages out to all clients that are connected and listening on a particular identifier. Again, in the same way as Quantlab would be triggered by a tick from a quote in the real-time feed from Reuters or Bloomberg.

```
vector(string) v_chat;

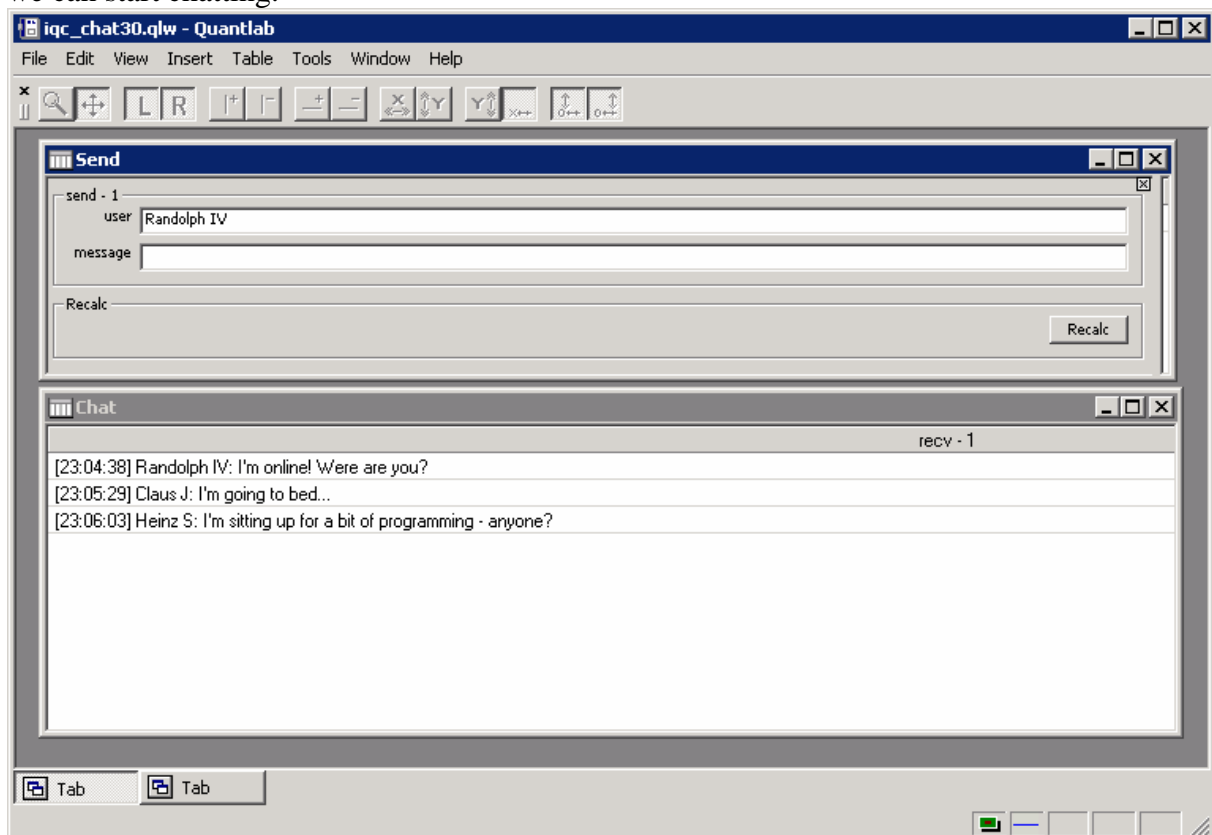
out vector(string) recv()
{
    push_back(v_chat, strcat([ "[",
        realtime_str("chat", "time", "IQC"), "]" , realtime_str("chat", "user", "IQC"), ": ",
        realtime_str("chat", "message", "IQC") ]));

    return v_chat;
}
```

First we have created a global variable (locally in the workspace) called v_chat. This vector of strings will hold all our received messages from the chat.

Second we create a function “recv()” that will execute at any time when the chat identifier has an updated data in it. The push_back function will just add another concatenated string into the v_chat variable. The function realtime_str() is a generic function that can be used to listen to realtime information streaming into Quantlab. It takes three arguments; the iqc identifier, the field identifier, and the name of the feed.

We can now attach both the send and receive functions to two tables in the user interface and we can start chatting.



It works! And the colleagues have already started pushing messages of their own ...

Example of feeding some market-maker corp spreads

We will create a mini-workspace with a table where the market maker can do manual input for three corporate bond spreads. Then we will create a user workspace that will price these bonds in terms of a base curve plus the spread published by the market maker.

```
vector(string) v_instr_name = ['CORP_BBB_1Y','CORP_BBB_5Y','CORP_BBB_10Y'];  
  
out void publish_spread(out vector(number) v_spread) {  
  
    for(i:0,v_size(v_spread)-1)  
        iqc_publish('IQC', v_instr_name[i], ['mid'], str([v_spread[i]]));  
  
}
```

Above is the code for the publishing part of the exercise. We place our three instrument names in a global variable. Then we create a function with an “out” vector as input argument. When the function is attached to a table the v_spread vector will be available for input by the user.

The loop will for each spread in the vector publish an iqc identifier and for each of these identifiers a mid quote.

In the second part we create some subscription code that will use the published spreads.

```
out vector(point_number) yields(curve_name base_c, date d, quote_side q){  
  
    vector(number) v_maturity = [1,5,10];  
  
    fit_result f = bootstrap(curve(base_c, d, q));  
  
    vector(number) zero_yields = f.zero_rate(0,v_maturity,RT_CONT);  
  
    vector(number) s = str_to_number(realtime_str(v_instr_name, 'mid','IQC'))/10000;  
  
    return point(v_maturity,(zero_yields + s)*100) ;  
  
}
```

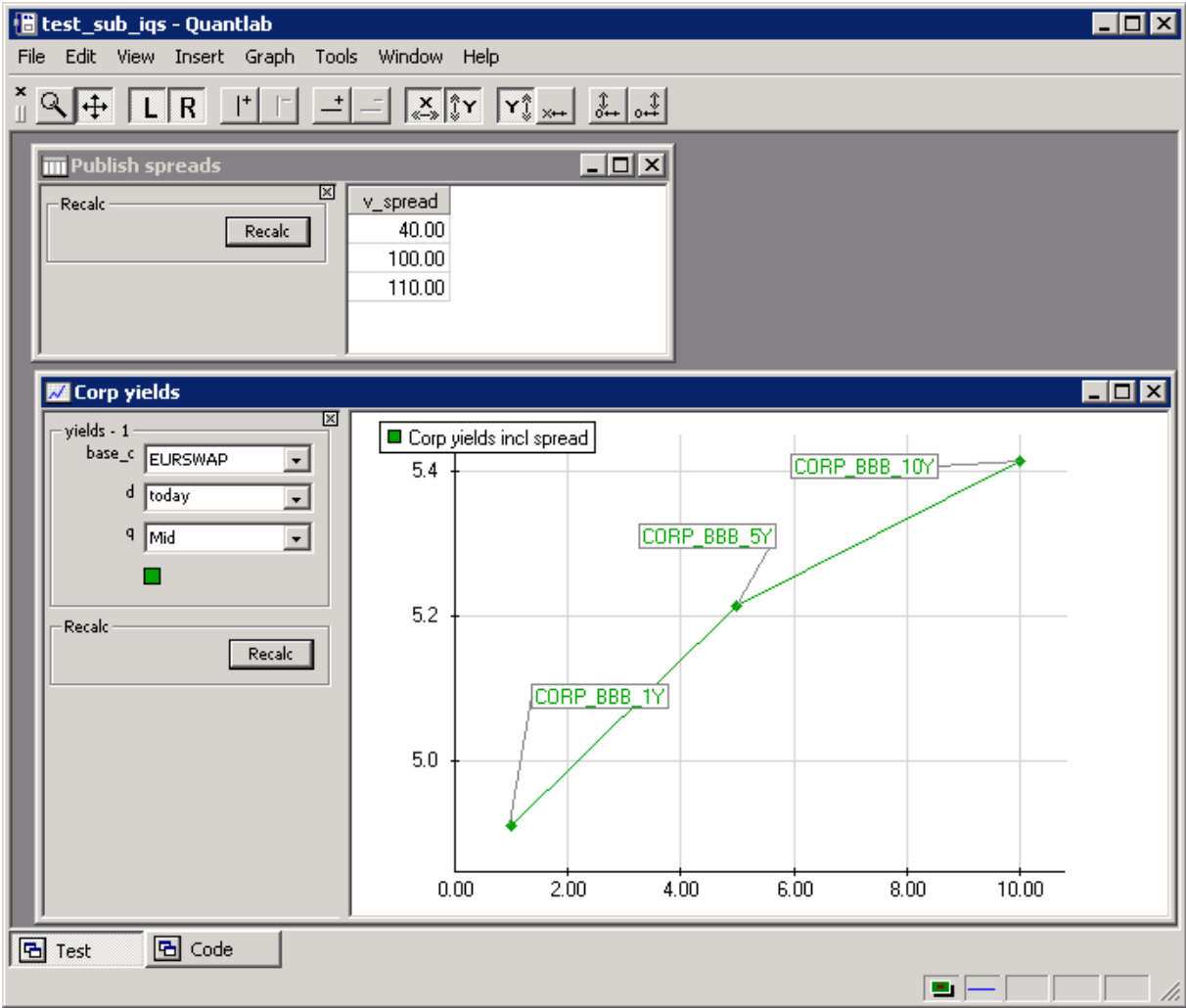
We have a function that will return a vector of points that we can attach to a graph. As input to our function we will allow the user to choose the base curve to price the bonds from. We will also allow the user to choose for which date to take the market quotes for the base curve as well as the quote side.

The base curve will be stripped from coupons to a zero coupon curve before we use it for pricing. We have chosen the bootstrap method. From the fitted curve we extract the zero yields for the maturities of our corporate bonds.

We then subscribe to the published corporate bond spreads using the realtime_str() function and divide the basis points with 10000.

It is now easy to return the three bonds zero yields as the sum of the base curve and the spreads.

Let's look at the workspace when we have attached the functions to a table and a graph.



For every time the market maker updates any of the spreads in the spread vector in his Quantlab workspace, the pricing will immediately be pushed to all other users listening to these iqc identifiers.