QUICKCHANNEL

fxbluelabs.com

1. Overview of QuickChannel

1.1 Purpose of QuickChannel

QuickChannel is a developer library for sending messages between EAs/scripts running in different instances of MT4 on the same computer (or within a single copy of MT4). Typical uses include trade duplication between different MT4 accounts, and price comparison in order to detect arbitrage opportunities.

QuickChannel can also be used to communicate between MT4 and any other application which allows DLL imports, including MT5 (32-bit or 64-bit).

QuickChannel can deliver messages in less than 1 millisecond, as illustrated by the log included with the example code. (However, the timing is very dependent on MT4's internal load. If a copy of MT4 is running more than one EA then the delivery time via QuickChannel will more typically be between 3 and 10 milliseconds.)

1.2 Licence

Unless otherwise specified in writing by FX Blue, this software is licensed for personal use only. You are not allowed to distribute it to other people, regardless of whether or not money is changing hands. You only have a licence to use QuickChannel if you, personally, downloaded it from our website.

The software is provided entirely at the user's risk, and you should check it thoroughly before deploying it on computers trading live money. FX Blue accepts no responsibility whatsoever for any loss of any kind caused by installation or use of the software.

2. Using QuickChannel

2.1 Installing QuickChannel

QuickChannel consists of a single DLL file: FXBlueQuickChannel.dll. This either needs to be copied into the experts\libraries directory of each MT4 installation which is going to use QuickChannel, or copied into one central location such as the System32 directory.

(There is also a 64-bit version of QuickChannel, for use with non-MT4 code. You can simply ignore this if you only want to use QuickChannel with MT4.)

MT4 EAs or scripts which use QuickChannel need to have their "Allow DLL imports" setting turned on when adding them to a chart.

2.2 Overview of communication via QuickChannel

EAs/scripts communicate with each other by using "channel names". In other words, the sender specifies a channel name such as EURUSDArbitrage when sending a message, and the receiver collects messages by asking for all the pending messages for EURUSDArbitrage.

- You can create an unlimited number of different channels.
- An EA can send messages down multiple channels, or receive from multiple channels.
- An EA can be both a sender and a receiver, i.e. can send on one channel and receive on another.
- Multiple EAs can send messages down the same channel.
- **But there should only be one receiver running on a channel**. If two EAs are both trying to receive messages on the same channel, then they will each get some of the messages (or one of the EAs will get all of them and the other will get none of them).

For example, let's say that you want to broadcast prices from one copy of MT4 to another, so that the receiver can look for arbitrage opportunities. You could do this as follows:

In the sending copy of MT4, you have an EA which you run on multiple charts for EURUSD, GBPUSD, USDJPY etc. These EAs simply send a message on each tick, reporting their current ask and bid price. All the EAs send down the same channel, e.g. "Arbitrage1".

In the receiving copy of MT4 you run **one** EA which collects messages on the "Arbitrage1" channel, and therefore this single receiving EA sees all the prices being sent from all the EAs in the sender copy of MT4.

This scenario is demonstrated by the example MQ4 code supplied with QuickChannel.

If you needed the receiver to be able to send messages back to the other copy of MT4, then you would simply use a second channel, e.g. "Arbitrage2". For example, the receiver might detect an arbitrage opportunity, buy EURUSD, and send a message back to the other copy of MT4 telling it to sell EURUSD. In the first copy of MT4 which sends the prices, you would run an additional EA which listened for messages on the "Arbitrage2" channel and acted on any instructions to place orders.

2.3 Overview of sending messages

An EA/script which sends messages via QuickChannel does three things:

- It opens a channel using the QC_StartSender() function.
- It sends one or more messages down the channel using the QC_SendMessage() function.
- It releases the channel using the QC_ReleaseSender() function.

QC_StartSender() is typically called in the EA's OnInit() and QC_ReleaseSender() is typically called in OnDeinit().

2.4 Overview of receiving messages

The process for receiving messages is very similar to that for sending messages:

- The EA/script opens a channel using the QC_StartReceiver() function.
- It checks for new messages using the QC_GetMessages3() function.
- It releases the channel using the QC_ReleaseReceiver() function.

The only quirk is that QC_StartReceiver() needs the window handle of the EA/script's chart. This is not always available during OnInit(), and therefore QC_StartReceiver() should be called in OnTick(), not OnInit(), obviously with a check to make sure that QC_StartReceiver() is only called once.

Once an EA/script has called QC_StartReceiver(), it receives a tick whenever a new message is available on the channel it has opened. In other words, its OnTick() function is run as soon as a new message is available, regardless of whether there has actually been a new tick from the broker for the symbol on whose chart the EA/script is running.

For confirmation: **receivers do not need to run an infinite loop** such as the following to make sure that they receive messages immediately rather than on the next chart tick:

```
// This is NOT NECESSARY…
while (!IsStopped()) {
        QC_GetMessages3(…);
        Sleep(1);
}
```

You can also monitor the channel by setting up a timer, i.e. using EventSetMillisecondTimer() and then monitoring for new messages in the OnTimer() event.

2.5 Channel lifespan

A channel exists for as long as any EA has called QC_StartSender() or QC_StartReceiver() without calling QC_ReleaseSender() or QC_ReleaseReceiver().

Once the last sending or receiving EA has released the channel, then it ceases to exist, and any pending uncollected messages are lost.

2.6 Multiple pending messages

Message transmission is asynchronous. There can be multiple calls to QC_SendMessage() before the receiver calls QC_GetMessages().

QuickChannel can store up uncollected messages so that QC_GetMessages3() subsequently returns a list of messages rather than a single message. In this case, the receiver gets back the list of messages separated by tabs. The example receiver MQ4 code shows one way of processing such a tab-separated list.

Senders can choose whether their message is added to any pending list which is waiting for collection, or whether any existing pending message(s) should be discarded and replaced by the new message. In simple terms, the QC_SendMessage() function has a parameter which indicates whether or not the channel should be cleared.

There are two points worth noting:

- If multiple different EAs are sending down the same channel then in most cases you will not want them to clear the channel each time they call QC_SendMessage(). Otherwise, only the most recent message will be stored, from one out of the EAs, and other messages may be lost.
- A tab-separated list from QC_Messages() is in the same order that the messages were sent. For some purposes you may need to process the list in **reverse order**, watching out for older messages which have been superseded by a newer message.

2.7 Message format and size

Messages are simply text. The only restriction is that you should not use the tab character in messages, because QuickChannel uses this as a delimiter between multiple messages.

The maximum size of pending messages on a channel is 131068 bytes. Calls to QC_SendMessage() will fail if the channel is already full of uncollected messages. By extension, this also means that the maximum size of any single message is 131068 bytes.

If you want to send numeric data, e.g. prices, you simply send it as text and then convert the text back to numbers in the receiver.

Similarly, if you want timestamps on messages then you simply include them in the message yourself, and get the receiver to break up what it receives into the timestamp and the actual message being sent. For example, the sender could build a pipe-delimited message:

```
string strMsg = StringConcatenate(TimeLocal(), "|", "My message");
```

…and the receiver could then use something like StringSplit() to crack the message into its constituent parts:

```
string MessageParts[];
StringSplit(strMsg, "|", MessageParts);
int Timestamp = StrToInteger(MessageParts[0]);
string Text = MessageParts[1];
```

3. MT4 function reference

3.1 Versions of MT4

v600+ of MT4 uses Unicode strings; older versions of MT4 use Ansi strings. The difference depends on which version of MT4 your EA/script is **compiled with**, not which version of MT4 your EA/script then runs on.

Because the different versions of EAs use different types of string, you need to use different versions of some functions depending on whether you are compiling with v600+ or the older v500.

3.2 DLL imports

3.2.1 DLL imports for v600+ of MT4

```
#import "FXBlueQuickChannel.dll"
  int QC_StartSenderW(string);
  int QC_ReleaseSender(int);
  int QC_SendMessageW(int, string, int);

  int QC_StartReceiverW(string, int);
  int QC_ReleaseReceiver(int);

  int QC_GetMessages5W(int, uchar&[], int);

  int QC_CheckChannelW(string);
  int QC_ChannelHasReceiverW(string);
#import
```

3.2.2 DLL imports for v500 of MT4

```
#import "FXBlueQuickChannel.dll"
  int QC_StartSender(string);
  int QC_ReleaseSender(int);
  int QC_SendMessage(int, string, int);

  int QC_StartReceiver(string, int);
  int QC_ReleaseReceiver(int);

  int QC_GetMessages3(int, string & arr[], int);
```

```
    int QC_CheckChannel(string);
    int QC_ChannelHasReceiver(string);
  #import
```

3.3 QC_StartSender()

MT4 v600:	int QC_StartSenderW(string ChannelName);
MT4 v500:	int QC_StartSender(string ChannelName);

QC_StartSender() opens a channel for sending. It returns a handle (i.e. an identifier) for the channel which should be used in subsequent calls to QC_ReleaseSender() and QC_SendMessage(). You should always release channels when you have finished with them, typically during OnDeinit().

A return value of zero indicates that the channel could not be opened. This usually means that system resources are critically low.

An EA/script will typically call QC_StartSender() in its OnInit() function, and store the handle in a global variable for later use.

3.4 QC_ReleaseSender()

MT4 v500 or v600: int QC_ReleaseSender(int ChannelHandle);

QC_ReleaseSender() frees up a channel which was previously opened using QC_StartSender(). The parameter is the handle returned by QC_StartSender().

The function returns 1 if the channel was successfully released.

3.5 QC_SendMessage()

MT4 v600:	int QC_SendMessageW(int ChannelHandle, string & Message, int Flags);
MT4 v500:	int QC_SendMessage(int ChannelHandle, string Message, int Flags);

QC_SendMessage() sends a message down a channel. The first two parameters are simply the handle returned by QC_StartSender() and the text of the message to be sent.

The third parameter is a flag bit-mask, i.e. a combination of options as follows:

Value	Meaning
1	Clear any pending uncollected messages and replace with this new message
2	Ignore the message if there is no receiver on the channel

Therefore, a flag value of 3 means "clear any pending messages, and also ignore the message if no receiver is running".

A flag value of 0 means "add this message to any existing messages which are already waiting for collection, and store the message for future collection even if no receiver is running yet".

A return value of 1 from QC_SendMessage() means that the message was successfully added to the channel. A return value of -1 means that the message was ignored because flag 2 was used and no receiver is currently connected to the channel.

A return value of 0 indicates failure to add the message. There are a number of possible reasons for this:

- The handle parameter for the channel is not valid.
- The system is critically low on resources.
- The channel is full. It already has 65532 bytes of pending messages waiting for collection by the receiver.

3.6 QC_StartReceiver()

```
MT4 v600:    int QC_StartReceiverW(string ChannelName, int WindowHandle);
MT4 v500:    int QC_StartReceiver(string ChannelName, int WindowHandle);
```

QC_StartReceiver() opens a channel for receiving messages. It returns a handle (i.e. an identifier) for the channel which should be used in subsequent calls to QC_ReleaseReceiver() and QC_GetMessages(). You should always release channels when you have finished with them, typically in OnDeinit().

There should only be one receiver running on a channel. If necessary, you can check whether another EA has already called QC_StartReceiver() by using the QC_ChannelHasReceiver() function.

The first parameter for QC_StartReceiver() is the channel name to open. The second parameter is the window handle of the MT4 chart on which the EA/script is running. This can be obtained in MQ4 using WindowHandle(Symbol(), Period()). For example:

```
glbHandle = QC_StartReceiver("MyChannel", WindowHandle(Symbol(), Period()))
```

Please note that WindowHandle() may return zero during OnInit() if MT4 is starting up with the EA already running on chart. Therefore, you should call QC_StartReceiver() during OnTick(), not during OnInit() – and obviously make sure that you only call it once during OnTick(), rather than on every tick.

A return value of zero indicates that the channel could not be opened. This typically means that system resources are critically low.

An EA/script will usually store the handle from QC_StartReceiver() in a global variable for later use by QC_GetMessages() and QC_ReleaseReceiver().

3.7 QC_ReleaseReceiver()

MT4 v500 or v600: int QC_ReleaseReceiver(int ChannelHandle);

QC_ReleaseReceiver() frees up a channel which was previously opened using QC_StartReceiver(). The parameter is the handle returned by QC_StartReceiver().

The function returns 1 if the channel was successfully released.

3.8 Retrieving messages

Different techniques are needed for retrieving messages on v500 and v600+ of MT4.

3.8.1 Retrieving messages on v600+ of MT4: QC_GetMessages5W()

MT4 v600: int QC_GetMessages5W(int, uchar&[], int);

In order to retrieve messages on v600+ of MT4 you need to create a uchar[] array, pass that array to the QC_GetMessages5W() function, and then use MT4's CharArrayToString() function to convert the uchar[] array to a string.

QC_GetMessages5W() returns an integer which is 0 if there are no pending messages; -1 if an error occurs; and otherwise returns the length of the data which has been put into the uchar[] buffer.

For example, using a buffer size defined by a constant:

```
uchar buffer[];
ArrayResize(buffer, QC_BUFFER_SIZE);
int res = QC_GetMessages5W(glbHandle, buffer, QC_BUFFER_SIZE);
if (res > 0) {
    string strMsg = CharArrayToString(buffer, 0, res);
    Print(strMsg);
}
```

3.8.2 Retrieving messages on v500 of MT4: QC_GetMessages3()

MT4 v500: `string QC_GetMessages3(int ChannelHandle, string & StringArray[], int BufferSize);`

QC_GetMessages3() collects all the messages which have been added to the channel since the receiver last retrieved its messages. There can potentially be more than one message waiting, and QC_GetMessages3() returns a tab-separated list.

Messages are delivered in the order in which they were sent. Bear in mind that for some purposes you may need to process the messages in **reverse** order, ignoring older messages which have been superseded by a newer message. The QuickChannel_Example_Receiver.mq4 example shows one way of processing a tab-separated list.

You need to build a buffer to pass to the QC_GetMessages3() function, and the list of pending messages, if any, is then put into that buffer. The buffer must be passed as element zero of a string array.

QC_GetMessages3() can return the following integer values describing success/failure:

- 0: Successfully put a non-blank message into the buffer provided.
- 1: No pending messages.
- 2: Buffer is not large enough to hold all the pending messages: the BufferSize parameter is not large enough.

The recommended way of using QC_GetMessages3() is as follows, as illustrated by the example code in QuickChannel_Example_Receiver.mq4:

- Declare a global variable to hold a "template" for the buffer.
- In init(), create a buffer which is large enough to contain the longest possible message.
- When calling QC_GetMessages3(), use a copy of the buffer template. This is faster than building a buffer of the required size before each call to QC_GetMessages3().

Example code is as follows:

```
// Buffer "template" created in init(); the actual buffer used for calls to QC_GetMessages3()
// is created by copying this variable
extern string glbBuffer = "";

void init()
{
    …
    // Build a "buffer template" for later use, storing it in the global-level variable
    glbBuffer = "01234567";
    for (int i = 0; i < 12; i++) glbBuffer = StringConcatenate(glbBuffer, glbBuffer);
}


void start()
{
    …
    // Create a copy of the buffer template and read the message list into it
    string arrBuffer[1];
    arrBuffer[0] = StringConcatenate(glbReceiveBuffer, "");  // Use copy of buffer template
    int res = QC_GetMessages3(glbHandle, arrBuffer, StringLen(arrBuffer[0]));

    // If successful (res == 0), then the list of pending messages is now in arrBuffer[0]
}
```

3.8.3 Copying messages to a file

MT4 v600:	int QC_GetMessages2W(int ChannelHandle, string Filename);
MT4 v500:	int QC_GetMessages2(int ChannelHandle, string Filename);

QC_GetMessages2() writes the list of pending messages, if any, to the specified filename. The file can be anywhere on disk (subject to file permissions), and you need to pass the full filename rather than the sort of bare relative filename used for MT4's own file functions. For example, to write to a file called mymessages in the experts\files directory of MT4, you would use the following:

```
int result = QC_GetMessages2(glbHandle, StringConcatenate(TerminalPath(),
"\experts\files\mymessages.txt"));
```

QC_Messages2() overwrites the specified file if it already exists, and can return the following integer values describing success/failure:

- 0: Successfully written non-blank list of messages
- 1: No pending messages
- 2: Unable to create the specified file. The name may be invalid, or the file may already be open and locked for writing.
- 3: Opened the file, but unable to write to it.

3.9 QC_CheckChannel()

MT4 v600: int QC_CheckChannelW(string ChannelName);
MT4 v500: int QC_CheckChannel(string ChannelName);

QC_CheckChannel() returns information about a channel. The possible return values are as follows:

Return	Meaning
-2	Error checking the channel. System is critically low on resources.
-1	Channel does not exist, i.e. has not already been opened by a sender or receiver using QC_StartSender() or QC_StartReceiver()
0	Channel is open, and there are no messages waiting for collection by a receiver
>0	Any value greater than zero indicates that there are pending messages waiting for collection by a receiver. The return value is the total length of the messages waiting for collection

3.10 QC_ChannelHasReceiver()

MT4 v600: int QC_ChannelHasReceiverW(string ChannelName);
MT4 v500: int QC_ChannelHasReceiver(string ChannelName);

QC_ChannelHasReceiver() checks whether a receiver is currently running on a channel, i.e. whether there is an EA/script which has called QC_StartReceiver(). The possible return values are as follows:

Return	Meaning
-2	Error checking the channel. System is critically low on resources.
-1	Channel does not exist at all, i.e. has not already been opened by a sender or receiver using QC_StartSender() or QC_StartReceiver()
0	Channel has at least one sender, but no receiver
1	Channel has a receiver

4. Use of QuickChannel in non-MT4 code

Although QuickChannel is principally designed for use with MT4, it can also be used in any other programming environment which allows DLL calls. For example, you can use it to send from MT4 to a receiver application written in C#, or vice versa. You can also use it to communicate between MT4 and MT5.

4.1 32-bit and 64-bit versions

The standard version of QuickChannel, for use with MT4, is a 32-bit library because MT4 is a 32-bit application.

However, there is also a 64-bit version of QuickChannel (FXBlueQuickChannel64.dll) for use with 64-bit applications, e.g. MT5.

An application which is using the 32-bit version **can** communicate with an application which is using the 64-bit version.

4.2 Collecting messages in non-MT4 code

In most programming environments, e.g. C#, it should be possible to use functions such as QC_GetMessages5W(), but there is also a QC_GetMessages4() function which returns a Windows BSTR (a managed Unicode string) and will be simpler to use in many environments including .NET.

```
BSTR QC_GetMessages4(IntPtr ChannelHandle);
```

Please see below for an example import of QC_GetMessages4() in C#.

4.2.1 Collecting messages in VBA/VB6

The QC_GetMessages5W() function can also be used in VBA code in Microsoft Excel (or in VB6). You call it in the same way as a Win32 function such as GetWindowText() which copies a string value into a buffer. For example:

```
Dim Buffer As String, szBuffer As Long, res As Long
szBuffer = 100000
Buffer = Space$(szBuffer + 1)
res = QC_GetMessages5W(qchandle, Buffer, szBuffer)
```

```
If res > 0 Then
        Dim strResult As String
        strResult = Left(Buffer, res)
        MsgBox strResult
ElseIf res = 0 Then
   ' No messages
Else
   ' Error, e.g. buffer too short
End If
```

4.3 Function imports in non-MT4 programming environments

You will need to translate the function definitions for MT4 into the corresponding definitions for your programming environment such as C# or VB.NET. Please note the following points:

- In the 64-bit version of QuickChannel, channel handles are 8 bytes long. Therefore, in 64-bit MT5 these should be declared as long rather than int, and in .NET you should use IntPtr.
- In 64-bit code, the window handle expected by QC_StartReceiver() is also 8 bytes long rather than 4 bytes long.
- If you are using VBA in Microsoft Excel, then please note that the standard installation of Microsoft Office is 32-bit even on a 64-bit version of Windows. Unless you have overridden the default installation, Office will be 32-bit and you should use the 32-bit version of QuickChannel.

4.3.1 Example imports for C#

The following imports are for 64-bit C# code. For 32-bit code, you simply change the DLL references from FXBlueQuickChannel64.dll to FXBlueQuickChannel.dll.

```
[DllImport("FXBlueQuickChannel64.dll",CharSet=CharSet.Unicode)]
public static extern IntPtr QC_StartSenderW(string ChannelName);

[DllImport("FXBlueQuickChannel64.dll",CharSet=CharSet.Unicode)]
public static extern int QC_SendMessageW(IntPtr ChannelHandle, string Msg, int Flags);

[DllImport("FXBlueQuickChannel64.dll")]
public static extern int QC_ReleaseSender(IntPtr ChannelHandle);

[DllImport("FXBlueQuickChannel64.dll", CharSet = CharSet.Unicode)]
public static extern IntPtr QC_StartReceiverW(string ChannelName, IntPtr WindowHandle);

[DllImport("FXBlueQuickChannel64.dll", CharSet = CharSet.Unicode)]
[return : MarshalAs(UnmanagedType.BStr)]
public static extern string QC_GetMessages4(IntPtr ChannelHandle);
```

```
[DllImport("FXBlueQuickChannel64.dll")]
public static extern int QC_ReleaseReceiver(IntPtr ChannelHandle);
```

4.3.2 Example imports for MT5

Code running on the 32-bit version of MT5 should use exactly the same DLL imports as MT4 v600.

For code running on 64-bit versions of MT5, you need to use the 64-bit version of the QuickChannel library, and to change the definitions of channel and window handles from 4 bytes (int) to 8 bytes (long):

```
#import "FXBlueQuickChannel64.dll"
  long QC_StartSenderW(string);
  int QC_ReleaseSender(long);
  int QC_SendMessageW(long, string, int);

  long QC_StartReceiverW(string, long);
  int QC_ReleaseReceiver(long);

  int QC_GetMessages5W(long, uchar&[], int);

  int QC_CheckChannelW(string);
  int QC_ChannelHasReceiverW(string);
#import
```

4.3.3 Example imports for VBA in Microsoft Excel

Please note that installations of Microsoft Office are usually 32-bit even on a 64-bit version of Windows, and therefore you should use the 32-bit version of QuickChannel.

```
Declare Function QC_StartSender lib "FXBlueQuickChannel.dll" (ByVal ChannelName As
String) As Long
Declare Function QC_ReleaseSender lib "FXBlueQuickChannel.dll" (ByVal ChannelHandle As
Long) As Long
Declare Function QC_SendMessage lib "FXBlueQuickChannel.dll" (ByVal ChannelHandle As
Long, ByVal Message As String, ByVal Flags As Long) As Long

Declare Function QC_StartReceiver lib "FXBlueQuickChannel.dll" (ByVal ChannelName As
String, ByVal WindowHandle As Long) As Long
Declare Function QC_ReleaseReceiver lib "FXBlueQuickChannel.dll" (ByVal ChannelHandle As
Long) As Long
```

```
Declare Function QC_GetMessages5W Lib "FXBlueQuickChannel.dll" (ByVal ChannelHandle As
Long, ByVal Buffer As String, ByVal BufferSize As Long) As Long

Declare Function QC_CheckChannel Lib "FXBlueQuickChannel.dll" (ByVal ChannelName As
String) As Long
Declare Function QC_ChannelHasReceiver Lib "FXBlueQuickChannel.dll" (ByVal ChannelName
As String) As Long
```

4.4 Event-driven message handling

In MT4, QuickChannel forces a call to MQL4's OnTick whenever a new message is available for collection by the receiver.

You can have similar event-driven processing in other environments by watching for Windows messages. When new data is available, QuickChannel posts an event to the window handle which you supply in QC_StartReceiver(). The Windows message details are as follows: msg = WM_USER (1024, 0x400), wparam = 1234, lparam = 5678.

For example, in C# you can trap these messages using the following code for the form or control whose handle you pass to QC_StartReceiver():

```
protected override void WndProc(ref Message m)
{
    if (m.Msg == 1024 && m.WParam.ToInt32() == 1234 && m.LParam.ToInt32() == 5678)
    {
        // A new message is available for collection using QC_GetMessages4()
    }
    else
    {
        // Call standard handling for this Windows message
        base.WndProc(ref m);
    }
}
```

Such processing of Windows messages is not available in MT5, and therefore you will simply have to watch for messages on a timed basis, e.g. using EventSetMillisecondTimer() and the OnTimer() event.

4.5 Window handles

You do not have to pass a valid window handle to QC_StartReceiver(). Therefore, you can still create a QuickChannel receiver in programming environments where a window handle is not available.

If you cannot pass a valid handle, you should instead pass the dummy value 1. The only side-effect of doing this is that QC_ChannelHasReceiver() will return 0 (because it checks the validity of the window handle associated with a channel).

5. Example MT4 code

QuickChannel is supplied with some example MQL4 code for EAs which send and receive messages. There are separate code examples for v500 and v600 of MT4.

QuickChannel_Example_Sender_v500.mq4 is an EA which simply sends the latest ask and prices for its chart on each tick. QuickChannel_Example_Receiver.mq4 is an EA which receives the messages from the sender(s), and logs them.

QuickChannel_Example_Receiver v500.mq4 also demonstrates one way of processing a tab-separated list of multiple messages returned by QC_GetMessages().

The v500 sender and receiver log their output to DbgView (which you can download from http://technet.microsoft.com/en-us/sysinternals/bb896647). This gives you precise timestamps for the sending and receiving, letting you check how quickly messages are being received. The QuickChannel package also includes an example log from DbgView showing average message transmission time of less than 1 millisecond.