

IQC User Guide

Version 2.48

December 14, 2020

Fully compatible with:

Windows, Linux, Mac OS

DTN IQFeed 5.0 - 6.1

MATLAB R2008b - R2020b

<https://Matlab-trader.com>



Table of Contents

DISCLAIMER	4
1 Introduction	5
2 Installation and licensing	6
2.1 Installing IQC	6
2.2 Licensing and activation	7
2.3 Switching activated computers	9
2.4 Updating the installed version	9
3 Using IQC	10
3.1 General usage	10
3.2 Common request properties	13
3.3 Blocking & non-blocking modes	13
3.4 Common causes of confusion	14
3.5 Returned data format	16
3.6 Run-time performance	18
4 Querying the latest market data	22
4.1 Snapshot (top of book) quotes	22
4.2 Fundamental information	31
4.3 Interval bars	36
4.4 Market depth (Level 2)	40
4.5 Greeks, fair value, and implied volatility	44
4.6 Market summary data and scanner	49
5 Historical and intra-day data	56
5.1 Daily data	56
5.2 Weekly data	62
5.3 Monthly data	65
5.4 Interval data	68
5.5 Tick data	75
5.6 Market summary data and scanner	81
6 Streaming data	83
6.1 Streaming quotes	83
6.2 Regional updates	89
6.3 Interval bars	93
6.4 Market depth (Level 2)	100
7 News	104
7.1 Configuration	104
7.2 Story headlines	105
7.3 Story text	109
7.4 Story count	111
7.5 Streaming news headlines	113
8 Lookup of symbols and codes	117
8.1 Symbols lookup	117
8.2 Options/futures chain	122
8.3 Markets lookup	127
8.4 Security types lookup	129
8.5 SIC codes lookup	131
8.6 NAICS codes lookup	133
8.7 Trade condition codes lookup	135

9 Connection, administration and other special commands	137
9.1 Connecting & disconnecting from IQFeed	137
9.2 Server time	140
9.3 Client stats	141
9.4 Sending a custom command to IQFeed	144
9.5 Modifying IQFeed's registry settings	145
10 Attaching user callbacks to IQFeed messages	146
10.1 Processing IQFeed messages in IQC	146
10.2 Run-time performance implications	150
10.3 Usage example – using callbacks to parse options/futures chains	151
10.4 Usage example – using callbacks for realtime quotes GUI updates	152
10.5 Usage example – using callbacks for realtime order-book GUI updates	153
11 Alerts	156
11.1 General Usage	156
11.2 Alert Configuration	158
11.3 Alerts Query	162
11.4 Alert Editing or Deletion	162
12 Messages and logging	163
12.1 IQC messages	163
12.2 IQFeed logging.....	165
13 Frequently-asked questions (FAQ)	168
14 Troubleshooting	171
15 Professional services	173
15.1 Sample program screenshots	174
15.2 About the author	177
16 Spread the word!	178
Appendix A – online resources	180
Appendix B – change log	181
B.1 Complete change log (functional + documentation)	181
B.2 Functional change log (excluding documentation changes)	196

DISCLAIMER

THIS SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND/OR NONINFRINGEMENT.

THIS SOFTWARE IS NOT OFFICIALLY APPROVED OR ENDORSED BY ANY REGULATORY, GOVERNING OR COMMERCIAL BODY, INCLUDING SEC, FINRA, MATHWORKS AND/OR DTN.

MUCH EFFORT WAS INVESTED TO ENSURE THE CORRECTNESS, ACCURACY AND USEFULNESS OF THE INFORMATION PRESENTED IN THIS DOCUMENT AND THE SOFTWARE. HOWEVER, THERE IS NEITHER A GUARANTEE THAT THE INFORMATION IS COMPLETE OR ERROR-FREE, NOR THAT IT MEETS THE USER’S NEEDS. THE AUTHOR AND COPYRIGHT HOLDERS TAKE ABSOLUTELY NO RESPONSIBILITY FOR POSSIBLE CONSEQUENCES DUE TO THIS DOCUMENT OR USE OF THE SOFTWARE.

THE FUNCTIONALITY OF THE SOFTWARE DEPENDS, IN PART, ON THE FUNCTIONALITY OF OTHER SOFTWARE, HARDWARE, SYSTEMS AND SERVICES BEYOND OUR CONTROL. SUCH EXTERNAL COMPONENTS MAY CHANGE OR STOP TO FUNCTION AT ANY TIME, WITHOUT PRIOR NOTICE AND WITHOUT OUR CONTROL. THEREFORE, THERE CAN BE NO ASSURANCE THAT THE SOFTWARE WOULD WORK, AS EXPECTED OR AT ALL, AT ANY GIVEN TIME.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES, LOSS, OR OTHER LIABILITY, WHETHER IN ACTION OF CONTRACT OR OTHERWISE, ARISING FROM, OUT OF, OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE, REGARDLESS OF FORM OF CLAIM OR WHETHER THE AUTHORS WERE ADVISED OF SUCH LIABILITIES.

WHEN USING THIS DOCUMENT AND SOFTWARE, USERS MUST VERIFY THE BEHAVIOR CAREFULLY ON THEIR SYSTEM BEFORE USING THE SAME FUNCTIONALITY FOR LIVE TRADES. USERS SHOULD EITHER USE THIS DOCUMENT AND SOFTWARE AT THEIR OWN RISK, OR NOT AT ALL.

ALL TRADING SYMBOLS AND TRADING ORDERS DISPLAYED IN THE DOCUMENTATION ARE FOR ILLUSTRATIVE PURPOSES ONLY AND ARE NOT INTENDED TO PORTRAY A TRADING RECOMMENDATION.

1 Introduction

DTN provides financial data-feed services via its IQFeed service (www.iqfeed.net). IQFeed customers use its services using a specialized application (“*client*”) that can be installed on the user’s computer.¹ User programs can interface to IQFeed’s client application in order to retrieve market data from the IQFeed servers.

Matlab is a programming platform that is widely-used in the financial sector. Matlab enables users to quickly analyze data, display results in graphs or interactive user interfaces, and to develop decision-support and automated trading programs.

IQFeed does not come with a Matlab API connector. This is the role of *IQC*. *IQC* is a program that runs in Matlab and connects it to IQFeed.

IQC provides a seamless Matlab interface to IQFeed data and functionality, via easy-to-use Matlab commands. Users avoid the complexities of IQFeed’s API integration.

IQC consists of three software components (in addition to this User Guide):

1. A Java package (*IQC.jar*) that connects Matlab to IQFeed’s client application.
2. A Matlab function (*IQC.p*) that provides IQFeed’s data in an easy-to-use manner, without needing to know anything about the underlying connector.
3. A Matlab file (*IQC.m*) that serves as a help file. This file contains no code, just the help comment; the code itself is contained in the two other software components. The help text is displayed when you run Matlab’s `help` function.

IQFeed queries (for trades and tick quotes, historical data, market info etc.) can be initiated with simple one-line Matlab code, using the Matlab function (*IQC.p*).

Users can easily attach Matlab code (callbacks) to incoming IQFeed messages. This enables users to trigger special operations (for example, adding an entry in an Excel file, sending an email or text message, sending a trade order to an OMS application) whenever a certain condition is detected, for example if a specified price is reached.

This document explains how to install and use *IQC*. Depending on the date that you installed *IQC*, your version may be missing some features discussed in this document. You can always download the latest *IQC* version from <http://IQC.net/files/IQC.zip>

¹ *IQConnect.exe* on Windows, *IQFeed application* on MacOS, or ran as a Windows app on Mac/Linux using Parallels/Wine. Note: some MacOS users have reported problems with the “native” app (which is basically just a bottled Wine installation) compared to a standard Wine install. This is a pure IQFeed/Mac issue, and not an *IQC* one; using Wine seems to solve it. In any case, only the IQFeed client needs to run under Wine - Matlab itself can run natively, but note that certain Windows-only functionality (such as discussed in §9.5 and §12.2) will not work in native mode, only under Parallels/Wine.

2 Installation and licensing

2.1 Installing IQC

IQC requires the following in order to work:

1. An active account at DTN IQFeed
2. An installation of the IQFeed client (*IQConnect*)
3. An installation of Matlab R2008a or a newer release

Installing *IQC* is simple:

1. Unzip the downloaded *zip* file in this local folder.
2. Add the local folder to your Matlab path using the path tool (run the Matlab `pathtool` command, or in the Matlab Desktop's toolstrip, click HOME / ENVIRONMENT / Set path... and save). The folder needs to be in your Matlab path whenever you run *IQC*.
3. If you are running the Production (non-trial) version of *IQC*, you will need to activate your license at this point. When you purchase your license you will be given specific instructions for this.
4. Ensure that the IQFeed client is working and can be used to log-in to IQFeed.⁴
5. You can now run *IQC* within Matlab. To verify that *IQC* is properly installed, retrieve the latest IQFeed server time, as follows (see §9.2):⁵

```
>> t = IQC('time');
```

6. You can query the installed version using *IQC*'s 'version' action, as follows:

```
>> disp(IQC('version'))  
Version: 1.0  
Release: '23-Feb-2018'  
License: 'Professional'  
Expiry: '16-Jun-2018'
```

2.2 Licensing and activation

IQC's license uses an activation that is specific to the installed computer. This uses a unique fingerprint hash-code that is reported by the Operating System, which includes the Windows ID (on Windows systems), computer name, and the list of hardware MAC addresses used by the computer.

Once the computer's license is activated, the activation key is stored on the *IQC* webserver. This activation key automatically validates online whenever *IQC* connects to IQFeed (i.e., at the beginning of an IQFeed session), and once every few hours while it is connected. Validating the license online only takes a second or two. Since it is only done at the initial connection to the IQFeed client and once every few hours, it does not affect *IQC*'s run-time performance. If you have a special concern regarding the online activation, please contact us for clarifications.

A corollary of the computer fingerprint is that whenever you modify components that affect the fingerprint, *IQC* will stop working. This could happen if you reinstall the operating system (OS), modify the computer name, change network adapters (e.g., switch between wifi/cellular/wired connection, or use a new USB networking device), modify MAC addresses, or use software that creates dynamic MACs. In such cases, you might see an error message such as the following when you try to use *IQC*:

```
Error using IQC
IQC is not activated on this computer.
```

Some additional information may be presented to help you diagnose the problem.

To fix such cases, simply revert back to the original hardware/networking configuration, and then *IQC* will resume working. If you wish to make the configuration change permanent, you can contact us for an activation switch to the new configuration (see the following section (§2.3) for details).

Computer fingerprints are typically stable, and are not supposed to change dynamically. However, some software programs (especially on MacOS, but also sometimes on Windows) create dynamic MAC addresses and/or dynamically modify the computer name (hostname). This may then be reflected in the OS-reported fingerprint, possibly causing *IQC* to stop working. The solution is to find a way to keep the fingerprint components static, with the same values as the activated fingerprint.⁶ You can determine the nature of the OS-reported fingerprint as follows:

```
>> IQC('license', 'debug', 1)
```

Using this command, you can determine which fingerprint component has changed and take the appropriate action to fix it so that the reported fingerprint will match the activated fingerprint. If you decide that the fingerprint changes are permanent, contact us to change the activated fingerprint (see §2.3).

A short time before your license term is over, you will start to see a notification message in your Matlab console (Command Window) alerting you about this:

```
*** Your IQC license will expire in 3 days (10-Mar-2018).
*** To extend your license please email info@IQC.net
```

⁶ For example, the computer's name can be set using the OS *hostname* command, or the following method on Mac OS: <https://knowledge.autodesk.com/support/smoke/learn-explore/caas/sfdcarticles/sfdcarticles/Setting-the-Mac-hostname-or-computer-name-from-the-terminal.html>

This informational message will only appear during the initial connection to IQFeed, so it will not affect your regular trading session.

When the license term is over, *IQC* will stop working and display an error message:

```
*** Your IQC license has expired on 10-Mar-2018.  
*** To extend your license please email info@IQC.net
```

You can always renew or extend your license using the payment links on <http://IQC.net> or <https://UndocumentedMatlab.com/IQC>. If you wish to be independent of such renewals, you can select a discounted long-term license.

You can query the installed version using *IQC*'s 'version' action, as follows:

```
>> data = IQC('version')  
data =  
    Version: 1.0  
    Release: '23-Feb-2018'  
    License: 'Professional'  
    Expiry: '16-Jun-2018'
```

Multiple *IQC* license options are available for purchase. Longer license terms are naturally more cost-effective than shorter ones. At the end of any license term you can decide to renew the same term, or purchase any other term:

- **Multi-year** license: these discounted long-term licenses will work for a longer duration than the standard license year without requiring a renewal, as long as IQFeed continues to provide its API service and your environment remains stable.
- **Volume (multi-computer)** license: the same license as for a single computer, but when you purchase multiple licenses at once, you get a volume discount.
- **Site** license: enables to run *IQC* on an unlimited number of computers within the same Windows Domain. This license does not require end-user activation, only a single centralized activation. It supports cloud deployment, where computer hardware fingerprints (but not the domain) often change.
- **Deployment (compiled or OEM)** license: enables to use *IQC* within a compiled program that runs on an unlimited number of computers, in any site or domain. This license does not require any end-user activation, only a single centralized activation of the parent executable to which the license is tied.
- **Source-code** license: unlimited in duration, can be installed on an unlimited number of computers within the organization, and requires no activation. This license requires signing a dedicated NDA (non-disclosure agreement).
- **Bundle** license: a discounted bundle of licenses for *IQC* and *IB-Matlab* (the InteractiveBrokers-Matlab connector). The combination of IB+IQFeed+Matlab is quite common in trading systems.

2.3 Switching activated computers

You can switch the *IQC* license activation between computers or computer hardware configurations (i.e., fingerprint hash-code) whenever you purchase a license renewal. For license terms of 1 year or longer, up to 2 activation switches per year are also included, at no extra cost. A handling fee will be incurred for other re-activations.

In order to change the activation fingerprint, simply email us the new configuration's fingerprint and we will make the switch on IQC's activation server.



Activation switches can take up to two business days to process, but typically complete within a few hours during European business hours. You will receive a confirmation email when the activation switch is complete.

3 Using IQC

3.1 General usage

IQC uses the IQFeed client ⁷ to connect to the IQFeed server. If an active IQFeed client is not detected, *IQC* will automatically attempt to start the IQFeed client and to connect to it. Note that this may not work for some IQFeed client installations. You can always start the IQFeed client manually, before running *IQC*. In any case, if an IQFeed connection is unsuccessful, *IQC* will error.

IQC's Matlab wrapper function is called *IQC*, contained within the *IQC.p* file. Its accompanying *IQC.m* file provides basic usage documentation using standard Matlab syntax, e.g.:

```
>> help('IQC')
>> help IQC      % equivalent alternative
>> doc IQC
```

The *IQC* function accepts a variable number of input parameters, and returns data in a single output argument, with an optional errorMsg output. The general syntax is:

```
>> [data, errorMsg] = IQC(action, parameters);
```

where:

- *data* is the output value. If this output value is requested, then Matlab processing will block data until the result is available; if the output *data* is not requested then the Matlab processing will proceed immediately (non-blocking) – the IQFeed data will stream asynchronously (see below).
- *errorMsg* is the latest error message that was reported (if any); see §3.5.
- *action* is a string that denotes the requested query type (mandatory input).
- *parameters* can be specified, depending on the requested *action*. There are several ways to specify *parameters*, which are described below.

For example:

```
>> data = IQC('time'); %'time' action (blocking), 0 parameters
>> IQC('quotes', 'Symbol', 'IBM'); %streaming 'quotes' action, 1 parameter
>> IQC('command', 'String', command, 'PortName', 'Admin'); %2 parameters
```

Note that when an output *data* is requested, *IQC* treats the request as blocking (synchronous), meaning that Matlab processing will wait for IQFeed's data (or a timeout) before proceeding with the next Matlab command. For example:

```
>> t = IQC('time'); % blocking until data is available
```

When an output *data* is *not* requested, *IQC* treats the request as streaming (non-blocking, a-synchronous) and Matlab processing will proceed immediately. This non-blocking mode is typically useful for sending IQFeed requests (for example, to start streaming trades/ticks), without waiting for a response from IQFeed. The streamed data is accumulated by *IQC* in the background, and can later be retrieved using the mechanism that is discussed in §6. Examples of such non-blocking commands:

```
>> IQC('quotes', 'Symbol', 'IBM'); %start non-blocking IBM quotes stream
>> IQC('command', 'String', command); %asynchronous/non-blocking command
```

⁷ *IQConnect.exe* on Windows, *IQFeed application* on MacOS. or ran as a Windows app on Mac/Linux using Parallels/Wine

Here are the `action` values recognized by *IQC*, in the Standard and Professional licenses; trial licenses have the full functionality of a Professional license:

Action	Description	User Guide Section(s)	Standard	Pro & trial
'version'	Display product version information	§2.1	Yes	Yes
'license'	Display the license fingerprint & activation key	§2.2	Yes	Yes
'update'	Update the <i>IQC</i> installation to the latest version	§2.4	Yes	Yes
'revert'	Update the <i>IQC</i> installation to a previous version	§2.4	Yes	Yes
'doc'	Display this User Guide in a separate window	-	Yes	Yes
'quotes'	Fetch quotes/trades information on a ticker	§4.1 , §6.1	Yes	Yes
'fundamental'	Fetch fundamental information on a ticker	§4.2	Yes	Yes
'intervalbars'	Fetch custom streaming interval bars on a ticker	§4.3 , §6.3	Yes	Yes
'marketdepth'	Fetch level 2 market depth information on a ticker	§4.4 , §6.4	-	Yes
'greeks'	Report option Greeks, fair value, implied volatility	§4.5	-	Yes
'history'	Fetch historical or intra-day data bars from IQFeed	§5	Yes	Yes
'summary'	Fetch historical market summary data from IQFeed	§5.6	-	Yes
'regional'	Fetch regional update information on a ticker	§6.2	-	Yes
'news'	Fetch news headlines or stories from IQFeed	§7	-	Yes
'lookup'	Fetch list of symbols/codes matching a set of criteria	§8	Yes	Yes
'chain'	Fetch futures/options chain matching a set of criteria	§8.2	-	Yes
'disconnect'	Disconnect <i>IQC</i> from IQFeed	§9.1	Yes	Yes
'reconnect'	Disconnect and then re-connect <i>IQC</i> to IQFeed	§9.1	Yes	Yes
'time'	Retrieve the latest IQFeed server & message times	§9.2	Yes	Yes
'stats'	Retrieve connection and network traffic statistics	§9.3	Yes	Yes
'command'	Send a custom command to IQFeed	§9.4	Yes	Yes
'registry'	Open Windows Registry Editor at IQFeed's settings	§9.5	Yes	Yes
'alert'	Alert the users upon IQFeed streaming events	§11	-	Yes
'log'	Control IQFeed logging of messages and events	§12.2	Yes	Yes

IQC accepts input parameters in several alternative formats, which are equivalent – you can use whichever format that you prefer:

- As name-value pairs – for example:


```
>> IQC('command', 'String', command, 'PortName', 'Admin'); %2 parameters
```
- As a Matlab struct, with parameters contained in corresponding struct fields e.g.:


```
>> params = []; % initialize
>> params.String = command;
>> params.PortName = 'Admin';
>> IQC('command', params);
```
- As a Matlab class, with parameters contained in corresponding class properties.
- As a Matlab table, with parameters contained in corresponding table variables.
- As field-separated rows in an Excel input file – for example:


```
>> IQC('command', 'C:\MyData\inputFile.xlsx');
```

where:

- Each column contains a separate parameter
- Row #1 contains the parameter names, and rows 2+ contain their corresponding values, one row per command
- All commands use the same `action` ('command' in this example)

	A	B
1	String	PortName
2	S,TIMESTAMPSOFF	Level1
3	S,CLIENTSTATS OFF	Admin
4	S,SET AUTOCONNECT,On	Admin

Each parameter must have an associated value. The value's data type depends on the specific parameter: it could be numeric, a string, a function handle etc. The definition of all the parameters and their expected data types is listed in the appropriate section in this User Guide that explains the usage for the associated `action`.

Note: if you specify parameters using a struct/class/table format, and then reuse this object for different *IQC* commands (by altering a few parameters), the entire set of parameters will be used, possibly including some leftover parameters from previous *IQC* commands. This may lead to unexpected results. For example:

```
% 1st IQC command - stop streaming timestamp messages every 1 second
>> params = []; % initialize
>> params.String = 'S,TIMESTAMPSOFF';
>> params.PortName = 'Level1';
>> IQC('command', params);

% 2nd IQC command - stop streaming client stats messages every 1 sec
>> params.String = 'S,CLIENTSTATS OFF'; %reuse existing params struct
>> IQC('command', params);

% 3rd IQC command - start streaming quotes messages for IBM
>> params.Symbol = 'IBM'; %reuse existing params struct
>> IQC('quotes', params);
```

In this example, the 2nd *IQC* command above will have no effect, because the **PortName** parameter in the `params` struct from the 1st *IQC* command will be reused in the 2nd command, sending it to the Level1 port, instead of to the Admin port. Similarly, the 3rd *IQC* command will result in a warning, because the 'quotes' action does not expect the **String** and **PortName** parameters that were carried over (reused) from the 2nd command. To avoid such unexpected results, it is therefore best to reset the object (`params=[]` for a struct) before preparing each *IQC* command.

IQC is quite tolerant of user input: parameter names (but generally not their values) are case-insensitive, parameter order does not matter, non-numeric parameter values can be specified as either char arrays ('abc') or strings ("abc"), and some of these can be shortened. For example, the following commands are all equivalent:

```
>> IQC('quotes', 'Symbol','IBM');
>> IQC('quotes', 'symbol','IBM');
>> IQC('Quotes', "Symbol","IBM");
>> IQC('Quotes', 'Symbol','IBM');
>> IQC('QUOTES', 'symbol',"IBM");
```

The full list of acceptable input parameters, and their expected values, is listed in the following sections, grouped by usage. If you specify an unexpected parameter, it will be ignored and a warning message will be displayed in the Matlab Command Window:

```
>> IQC('quotes', 'badName',1)
Warning: 'badName' is not a valid parameter for the 'quotes' action in IQC
```

When using *IQC*, there is no need to worry about connecting or disconnecting from the IQFeed client – *IQC* handles these activities automatically, without requiring user intervention. Users only need to ensure that the IQFeed client is active and logged-in when the *IQC* command is invoked in Matlab.

IQC reads data using the IQFeed account to which the IQFeed client is connected. In other words, the IQFeed account type is transparent to *IQC*: the only way to control which IQFeed data is available to *IQC* is to login to the IQFeed client using the appropriate username/password. Refer to §9.1 for additional details.

3.2 Common request properties

The following properties can be specified in *IQC*, with most actions:

Parameter	Data type	Default	Description
Symbol or Symbols ⁸	string	(none)	The asset symbol, as known by IQFeed. ⁹
Timeout	number	5.0	Max number of seconds (0-9000) to wait for data in a blocking request (0 means infinite).
Debug	logical	false or 0	If true or 1, additional information is displayed.
MsgParsingLevel	number	2	One of: <ul style="list-style-type: none"> • 2 – parse all the data in incoming IQFeed messages (default; most verbose, slowest) • 1 – do not parse lookup codes (e.g. trade condition, price formats, market id). The corresponding Description fields will either be missing, or contain empty strings. The codes can be parsed separately (see §8). • 0 – do not parse lookup code; also do not convert string data into numeric values (i.e. all data fields will remain strings: '3.14'). This is the fastest but least verbose option.
RaiseErrorMsgs	logical	true or 1 ¹⁰	If true or 1, IQFeed error messages raise a Matlab error in blocking (non-streaming) mode (see §12)
ProcessFunc	function handle	[]	Custom user callback function to process incoming IQFeed data messages (see §10).
NumOfEvents	integer	inf	The maximal number of messages to process.

Additional properties are request-specific and are listed below in the relevant sections. For example, the 'history' action has additional properties that control the parameters of the historic data request (start/end date, data type, etc.).

3.3 Blocking & non-blocking modes

Whenever you specify an output parameter in a call to *IQC*, the program will block until a response is available (i.e., a *synchronous* request). If no output parameter is specified, *IQC* will immediately return (non-blocking, *a-synchronous*) and additional Matlab commands can immediately be issued. This non-blocking mode is typically useful for sending IQFeed requests to start streaming data (for example, streaming trades/ticks or news headlines), without waiting for any response from IQFeed. The streamed data is accumulated by IQC in the background, and can later be retrieved using the mechanism that is discussed in §6. For example:

```
>> t = IQC('time'); % blocking until data is available
>> IQC('quotes', 'Symbol', 'IBM'); %start non-blocking IBM quotes stream
>> IQC('command', 'String', command); %asynchronous/non-blocking command
```

⁸ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization.

⁹ <https://iqfeed.net/symbolguide>

¹⁰ Using the 2nd (optional) output parameter of *IQC* implies a default value of false (0) for **RaiseErrorMsgs** (see §3.5)

3.4 Common causes of confusion

1. A common cause of error is specifying symbols incorrectly. IQFeed is very sensitive about this: if the specified symbol is invalid,¹¹ or if your account does not have the corresponding market subscription, IQFeed will report an error:

```
>> IQC('quotes', 'Symbol', 'xyz123')
Symbol 'XYZ123' was not found!
```

If the request is blocking, an error (exception) will be thrown (raised), which can be trapped and handled by the user, using a Matlab try-catch construct:

```
try
    data = IQC('fundamental', 'Symbol', 'xyz123');
catch err
    fprintf(2, 'Error: %s\n', err.message);
    % do some intelligent processing here...
end
```

IQFeed's website includes a detailed symbol-lookup search engine.¹² If you are still unsure about a symbol name, please contact IQFeed's customer support.

2. If any request parameter is invalid or if the request is not accepted by IQFeed, a run-time error will result, which can be trapped as shown above. For example:

```
IQC historic data query (EURGBP.FXCM) error: Unauthorized user
ID (your IQFeed account is not authorized for this data)
```

3. A common confusion source is specifying numeric values as strings or vice versa. For example, `IQC(..., 'Timeout', '10')` rather than `IQC(..., 'Timeout', 10)`. Each *IQC* parameter expects a value of a specific data type, as listed in the parameter tables in this user guide. *IQC* is sometimes smart enough to automatically convert to the correct data type, but you should not rely on this: it is better to always use the correct data type. Otherwise, Matlab might get confused when trying to interpret the string '10' as a number, and odd results might happen.
4. While most of *IQC*'s functionality is available in all license types, some actions/functionality are only available in the Professional *IQC* license:¹³
 - Parallelized queries (§3.6)
 - Customizable data fields in quotes data (§4.1, §6.1)
 - Level 2 market depth quotes (§4.4, §6.4, §10.5)
 - Option Greeks, Fair Value and Implied Volatility (§4.5)
 - Regional updates (§6.2)
 - News (§7)
 - Options/futures chain lookup (§8.2)
 - Alerts (§11)

If you have a Standard license and try to access Professional-only functionality, a run-time error will result:

```
>> data = IQC('news');
The 'news' action is not available in your Standard license of IQC, only
in the Professional license. Contact info@IQC.net to upgrade your license.
```

¹¹ For example, EURUSD.FXCM is a valid symbol, but EURUSD and USDEUR.FXCM are invalid

¹² <https://iqfeed.net/symbolguide>

¹³ A Standard license can be converted into a Professional license at any time; contact info@IQC.net for details.

5. IQFeed reports dates in different formats, depending on the specific query: either in the standard American mm/dd/yyyy format (for example: '01/29/2018'), or in yyymmdd format (for example: '2018-01-29' or '20180129 12:29:48'). Dates are usually reported as strings. In some cases, a corresponding Matlab datenum value is also reported, for example (§5.5, §6.1):

```

Symbol: 'IBM'
Timestamp: '2018-03-07 13:23:02.036440'
Datenum: 737126.557662458
...
Symbol: '@VX#'
LatestEventDenum: 737128.637260451
LatestEventTimestamp: '20180309 15:17:39'
...
```

Depending on the data field, the timestamp is either your local computer's time, or IQFeed servers (New York) time – **not** the exchange time. To get the exchange time, you would need to do the appropriate time-zone arithmetic.

6. By default, Matlab displays data in the console (“Command Window”) using “short” format, which displays numbers rounded to 4 digits after the decimal. The data actually has higher precision, so when you use it in a calculation the full precision is used, but this is simply not displayed in the console.

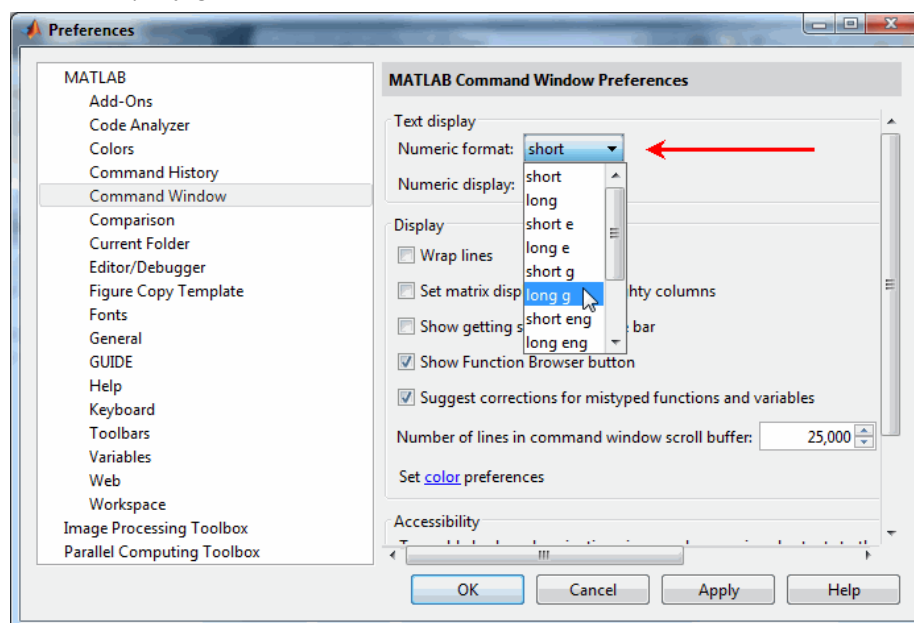


IQC does not truncate/round/modify the IQFeed data in any manner!

To display the full numeric precision in the Matlab console, change your Command Window's Numeric Format from “short” to “long” (or “long g”) in Matlab's Preferences window, or use the “format long” Matlab command:

```

>> data = IQC('quotes', 'symbol', 'ONLIB.X'); %overnight LIBOR rate
>> data.Close % short format (only 4 digits after decimal)
ans =
    1.4463
>> format long g % long format (full precision displayed)
>> data.Close
ans =
    1.44625
```



3.5 Returned data format

Many queries in *IQC* return their data in the form of a struct-array (a Matlab array of structs), for example (see §8.6):

```
>> data = IQC('lookup', 'DataType', 'NAICS')
data =
    1175x1 struct array with fields:
        id
    description

>> data(1)
ans =
        id: 111110
    description: 'Soybean Farming'

>> data(2)
ans =
        id: 111120
    description: 'Oilseed (except Soybean) Farming'
```

For various purposes (readability, maintainability, performance, usability), users may wish to modify this data structure. You can easily convert the data using Matlab's builtin functions `struct2cell()` (which converts the struct-array into a cell-array), or `struct2table()` (which converts the struct-array into a Matlab table object):

```
>> disp(struct2cell(data)')
    [111110]    'Soybean Farming'
    [111120]    'Oilseed (except Soybean) Farming'
    [111130]    'Dry Pea and Bean Farming'
    [111140]    'Wheat Farming'
    [111150]    'Corn Farming'
    [111160]    'Rice Farming'
    ...

>> disp(struct2table(data))
        id                description
    _____
    111110    'Soybean Farming'
    111120    'Oilseed (except Soybean) Farming'
    111130    'Dry Pea and Bean Farming'
    111140    'Wheat Farming'
    111150    'Corn Farming'
    111160    'Rice Farming'
    ...
```

Note that empty data cannot be converted using `struct2table()` or `struct2cell()`:

```
>> data = IQC('lookup', 'DataType', 'NAICS', 'Description', 'xyz')
data =
    []

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.

>> struct2table(data)
Error using struct2table (line 26)
S must be a scalar structure, or a structure array with one column or one row.
```

When requesting data for multiple symbols, *IQC* automatically tries to merge the data into an NxM array of structs (N data points for each of M symbols). If this is impossible (e.g. different number of data points per symbol), the results are reported as a cell array. For example:


```
>> data = IQC('history', 'Symbol','IBM,AAPL,T')
data =
    100x3 struct array with fields:
        Symbol
        Datestamp
        Datenum
        High
        Low
        Open
        Close
        PeriodVolume
        OpenInterest

>> data = IQC('history', 'Symbol','IBM,AAPL,T,@ESH20C120000')
data =
    1x4 cell array
    {100x1 struct} {100x1 struct} {100x1 struct} {65x1 struct}
```

A second, optional, output parameter of *IQC* returns the latest error message (if any):¹⁴

```
>> [data, errorMsg] = IQC('quotes', 'Symbol','IBM', 'Timeout',0.1)
data =
    []
errorMsg =
    'IQC timeout: either IQFeed has no data for this query, or the Timeout
    parameter should be set to a value larger than 0.1'
```

¹⁴ Using the 2nd (optional) output parameter of *IQC* implies a default value of false (0) for the **RaiseErrorMsgs** parameter.

3.6 Run-time performance

3.6.1 General considerations

IQC's standard processing has an overhead of 1-2 milliseconds per IQFeed message, depending on several factors:

- Message type/complexity – simple messages such as the periodic timestamp updates are simpler to process than complex messages (e.g. fundamental data).
- The **Debug** parameter (see §3.2) – A value of 1/ `true` is ~1 msec *slower* per message, compared to the default value of 0/`false` (depending on message type).
- The **MsgParsingLevel** parameter (§3.2) – A value of 0 is ~1 msec *faster* per message, compared to the default value of 2 (depending on message type).
- The **UseParallel** parameter (see below) enables query parallelization (*faster*).
- The **Fields** parameter (§4.1, §5.1-§5.5, §6.1) – fewer fields are *faster*.
- User-defined callbacks (§10) add their own processing time per message. See §10.2 for suggested ways to speed-up this callback processing overhead.
- Each active alert (§11) uses 1-2 msec per message (depending on alert type, and only for the alert's corresponding message type). If the alert action is triggered, then its processing time is added. For example, displaying a popup message might take 1 sec, and sending an email might take a few seconds.
- Computer capabilities – faster CPU and memory (RAM) enable faster processing, if your computer has enough physical memory to avoid swapping. Adding memory is typically much more cost-effective than upgrading the CPU.

This means that without any defined alert or user-specified callback, nor any other code running in the background (for example, a Matlab data analysis program), we can expect *IQC* to process up to 500-1000 IQFeed messages per second by default.

This is a relatively fast throughput, but if you stream real-time quotes for hundreds of liquid securities concurrently then you might reach this limit. When this happens, Matlab may be so bogged-down from the flood of incoming messages that it will become unresponsive, and you may need to restart *IQConnect* and/or Matlab.

Similarly, if you request a blocking (non-streaming) request with multiple data items (for example, thousands of historical data or news items), the query may take a while to process, requiring us to set a higher-than-default **Timeout** parameter value. For example, if you issue a blocking request for 20K data bars, IQFeed will send 20K data messages (one message per bar). If each of these messages takes 1-2 msec to process, the total processing time for the *IQC* query will be 20-40 secs.

When IQFeed is connected, it continuously sends messages to *IQC*: periodic “heartbeat” and status messages, and messages for any active streaming quotes or news events that you requested. These messages are automatically processed by *IQC* in the background, reducing the CPU time that is left available to process other *IQC* queries (e.g., a blocking historical data query) or Matlab analysis functions. It is therefore advisable to stop streaming IQFeed data when not needed, even if only temporarily.

IQFeed includes a built-in logging mechanism. Setting the logger to record fewer data items will improve run-time performance (see §12.2). IQFeed's log file can become very large very fast and reduce system performance if left unattended. It is intended to be used for troubleshooting purposes only and not on a constant basis. This is especially true if you log streaming data, large historic data, and/or Debug data.

Also, by default IQFeed stores the log file in the user's *Documents* folder, which might be automatically synced to OneDrive or Dropbox clouds. The log file can grow to several GBs per day, so such a sync could result in marked slowdown.¹⁵ To improve performance, disable this sync, or place the log file in a non-synced folder.

3.6.2 Paralellization

With the Professional and trial *IQC* licenses, you can use Matlab's Parallel Computing Toolbox to parallelize IQFeed queries. This can be done both externally (placing *IQC* commands in `parfor`/`spmd` blocks, so that they will run independently), and internally (for some *IQC* query types, using the **UseParallel** parameter). If you have the Standard *IQC* license, or if you do not have Matlab's Parallel Computing Toolbox, you can still run concurrent *IQC* commands in separate Matlab sessions, just not in the same session.

IQC automatically tries to parallelize queries when the **UseParallel** parameter value (default: `false`) is set to 1 or `true`. The list of parallelizable queries includes:

- Requests resulting in multiple blocking queries in a single *IQC* command (for example, historical data for multiple symbols or a date range – see §5)
- Requests for full news-story of all returned news headlines in a blocking query, using the **GetStory** parameter (see §7.2)
- Requests for fundamental/quotes data on all symbols in an options-chain or futures-chain, using the **WhatToShow** parameter (see §8.2)

When setting **UseParallel** to 1 or `true`, *IQC* will use parallel Matlab tasks (so-called 'headless workers' or 'labs') from the currently-active parallel pool created by the Parallel Computing Toolbox. If no pool is active, the default local pool is automatically started.

Here is a run-time example showing the effect of using a 4-worker pool to parallelize a news-story query, resulting in a 3.5x speedup (not 4x, due to parallelization overheads):

```
>> tic, data = IQC('news', 'DataType','headlines', 'MaxItems',100, ...
                  'GetStory',1); toc
Elapsed time is 56.311768 seconds.

>> parpool('local',4) % start 4 workers in parallel pool (optional)
>> tic, data = IQC('news', 'DataType','headlines', 'MaxItems',100, ...
                  'GetStory',1, 'UseParallel',1); toc
Elapsed time is 15.799185 seconds.
```

Depending on the number of CPU cores and available memory (RAM) on your computer, and the amount of work that the OS and other tasks as doing in the background, a pool of 14 workers (the maximum for IQFeed and *IQC*) can result in a speedup of up to 10-12x for parallelizable queries (such as historical data fetches), compared to the corresponding non-parallel (serial) query.

¹⁵ <http://forums.dtn.com/index.cfm?page=topic&topicID=5771>

IQC parallelization has several performance implications:

- Starting the parallel pool can take some time (a few seconds, up to a minute or two, depending on configuration). It is best to start the pool before any time-critical operations to avoid this startup time upon the first parallel query. Starting the pool (with an optional size) can be done using Matlab's `parpool` function.
- The default pool uses the same number of workers as the number of physical cores on your computer. This makes sense for CPU-intensive programs, but *IQC* queries are limited by I/O, not CPU. Therefore, unless you also use the parallel pool for CPU-intensive computations in your program, it makes sense to start a pool that has more workers than the number of CPU cores. You can configure your local cluster for this.¹⁶ Note that IQFeed limits the number of concurrent requests of some query types, limiting the effective parallel pool size.¹⁷ Larger worker pools use extra CPU and memory and degrade *IQC*'s overall performance. Different pool sizes are optimal for different queries; you can try using various pools, to find an optimal size for your specific queries and system.
- For any parallelizable *IQC* command, you can set a **MaxWorkers** parameter value that limits number of workers for that command (up to the pool size). When **MaxWorkers** < pool size, only **MaxWorkers** will be used to process the *IQC* command; the other workers will remain idle and consume no CPU.
- Changing the parallel pool size entails a restart of the entire pool, which could take quite some time. It is best to pre-allocate the maximal pool size that you will need (using the optional pool size argument of the `parpool` command, or by configuring the local cluster's default pool size). When you need less workers for a command (for example, only 15 workers in a history command), use the **MaxWorkers** parameter to limit workers without modifying the pool size.
- In addition to the workers startup time, each worker independently connects to IQFeed upon the first *IQC* command it encounters, taking a few extra seconds.
- It is only possible to parallelize workers on the local computer, not on other (distributed) computers in a grid/cluster/cloud. This is due to IQFeed/exchange limitations, which prohibit distribution of data to other computers.
- Due to parallelization overheads, inter-task memory transfers, and CPU task-switches (especially in a case of more workers than cores), speedup will always be smaller than the number of workers. The actual speedup will depend on query type and computer/OS configuration. Parallelization may even cause slowdown in some cases (e.g. quote queries, due to waiting for market events, not CPU).

Historical data queries have two parallelization variants:

- When requesting data for *multiple symbols*, the symbols are split across the available parallel workers (up to the specified **MaxWorkers** value or the active pool size, whichever is lower), one or more complete symbol query per worker. Therefore, if you query fewer symbols than workers, some workers will remain

¹⁶ <https://www.mathworks.com/help/distcomp/discover-clusters-and-use-cluster-profiles.html#f5-16540>

¹⁷ <http://forums.dtn.com/index.cfm?page=topic&topicID=5748> states that history queries are limited to 15 concurrent requests

idle (unused). In such a case, consider using a smaller parallel pool, with fewer workers, taking up less CPU and memory resources. Alternatively, run a serial for-loop over the symbols, parallelizing the date/time dimension (next bullet).

- When requesting data for just a *single symbol*, the requested date/time segment (**BeginDate** to **EndDate** etc.) is evenly split across the workers. You should always set **BeginDate** to the earliest time of actual available data: without a date/time segment (for example no **BeginDate**), parallelization is impossible so *IQC* uses slow serial mode. Likewise, when data is limited (for example, 1-sec intervals are only available for the past 180 days) and **BeginDate** is older than the earliest data, some workers will be idle, running actual work on fewer workers (slower).

3.6.3 Reported data-fields

Also in the Professional *IQC* license, you can customize the fields that *IQC* reports for market data quotes and historic data bars. The **Fields** parameter can be set to a cell-array of strings (`'Bid','Ask','Last'`), or a comma-separated string (`'Bid,Ask,Last'`). All subsequent queries of the same type will only report the requested fields. For example:

```
>> data = IQC('quotes', 'Symbol', 'AAPL', 'Fields', {'Bid', 'Ask', 'Last'})
>> data = IQC('quotes', 'Symbol', 'AAPL', 'Fields', 'Bid,Ask,Last') %equivalent
data =
    Symbol: 'AAPL'
         Bid: 222.71
         Ask: 222.91
         Last: 222.11
```

Note: the fewer fields that you request, the faster the required processing time, by both IQFeed and *IQC* (see §4.1, §5.1-§5.5, §6.1 for details). Requesting fewer fields (as in the example above, which only requested 3 quote fields) results in faster processing and reduced memory consumption. To improve the run-time and reduce memory usage it is advisable to request only those data fields that your program actually needs.

4 Querying the latest market data

4.1 Snapshot (top of book) quotes

We start with a simple example to retrieve the latest market information for Alphabet Inc. Class C, which trades using the GOOG symbol, using *IQC*'s 'quotes' action:

```
>> data = IQC('quotes', 'Symbol', 'GOOG')
data =

    Symbol: 'GOOG'
    Most_Recent_Trade: 1092.14
    Most_Recent_Trade_Size: 1
    Most_Recent_Trade_Time: '09:46:31.960276'
    Most_Recent_Trade_Market_Center: 25
    Total_Volume: 113677
    Bid: 1092.13
    Bid_Size: 100
    Ask: 1092.99
    Ask_Size: 100
    Open: 1099.22
    High: 1099.22
    Low: 1092.38
    Close: 1090.93
    Message_Contents: 'Cbaohlc'
    Message_Description: 'Last qualified trade; A bid update
    occurred; An ask update occurred; An open
    declaration occurred; A high declaration
    occurred; A low declaration occurred; A
    close declaration occurred'
    Most_Recent_Trade_Conditions: '3D87'
    Trade_Conditions_Description: 'Intramaket Sweep; Odd lot trade'
    Most_Recent_Market_Name: 'Direct Edge A (EDGA)'
```

As can be seen, the returned data object is a Matlab struct with self-explanatory fields.¹⁸ To access any specific field, use the standard Matlab dot-notation:

```
>> bidPrice = data.Bid; % = 1092.13 in this specific case
```

If the symbol is not currently trading, some fields return empty values:

```
>> data = IQC('quotes', 'Symbol', 'GOOG')
data =

    Symbol: 'GOOG'
    Most_Recent_Trade: 1078.99
    Most_Recent_Trade_Size: 1
    Most_Recent_Trade_Time: '19:58:47.052099'
    Most_Recent_Trade_Market_Center: 26
    Total_Volume: 0
    Bid: 1077.6
    Bid_Size: 100
    Ask: 1079.89
    Ask_Size: 200
    Open: []
    High: []
    Low: []
    Close: 1078.92
    Message_Contents: 'Cbav'
    Message_Description: 'Last qualified trade; A bid update
    occurred; An ask update occurred;
    A volume update occurred'
    Most_Recent_Trade_Conditions: '0517'
    Trade_Conditions_Description: 'Average Price Trade; Form-T Trade'
    Most_Recent_Market_Name: 'Direct Edge X (EDGX)'
```

¹⁸ The textual Description fields depend on the **MsgParsingLevel** parameter having a value of 2 or higher (see §3.2 and §8). The Bid and Ask fields are typically NBBO (National Best Bid and Offer) quotes.

In this example, the query was sent outside regular trading hours (on Sunday) so `Open`, `High` and `Low` are empty. As expected, the data indicates this was a “Form-T” trade.

Other fields may sometimes be empty. For example, overnight LIBOR rate (**Symbol**=`'ONLIB.X'`) reports empty `Bid`, `Ask`, `Most_Recent_Trade_Size` (and `Total_Volume=0`).

In rare cases, you might see invalid field values (e.g. 0), which may indicate a data error. *IQC* does not modify the data reported by IQFeed, so if you see this problem consistently for a certain security or exchange, please contact IQFeed’s support.

If you specify an incorrect security name or classification properties, or if you do not have the necessary market subscription, then no data is returned, and an error message is displayed (see discussion in §3.4).

```
>> IQC('quotes', 'Symbol', 'xyz123')
Symbol 'XYZ123' was not found!
```

You may request more than a single snapshot quote: To get the next N real-time quotes, specify the **NumOfEvents** parameter. The result is an array of structs in the same format as above (or an empty array if no data is available):¹⁹

```
>> data = IQC('quotes', 'Symbol', 'IBM', 'NumOfEvents', 4)
data =
    4×1 struct array with fields:
        Symbol
        Most_Recent_Trade
        Most_Recent_Trade_Size
        ...

>> data(1)
ans =

                Symbol: 'IBM'
        Most_Recent_Trade: 159.16
    Most_Recent_Trade_Size: 75
    Most_Recent_Trade_Time: '09:36:15.534201'
    Most_Recent_Trade_Market_Center: 24
                Total_Volume: 135267
                ...
```

Note that it is possible that not all the requested quotes will be received before *IQC*’s timeout (default value: 5 secs) returns the results:

```
>> data = IQC('quotes', 'Symbol', 'IBM', 'NumOfEvents', 4)
Warning: IQC timeout: only partial data is returned. Perhaps the Timeout
parameter should be set to a value larger than 5 or the NumOfEvents parameter
to a value smaller than 4
data =
    2×1 struct array with fields:
        Symbol
        Most_Recent_Trade
        Most_Recent_Trade_Size
        ...
```

To control the maximal duration that *IQC* will wait for the data, set the **Timeout** parameter. For example, to wait up to 60 secs to collect the next 4 upcoming quotes:

```
>> data = IQC('quotes', 'Symbol', 'IBM', 'NumOfEvents', 4, 'timeout', 60);
```

¹⁹ Some older versions of *IQC* returned a different form struct (the same as that reported by streaming quotes - §6.1). This was corrected to match the documentation starting in *IQC* version 2.00.

You can request quotes for multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('quotes', 'symbols',{'IBM','GOOG','AAPL'});
>> data = IQC('quotes', 'symbols','IBM:GOOG:AAPL'); % equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols:

```
data =
1x3 struct array with fields:
    Symbol
    Most_Recent_Trade
    Most_Recent_Trade_Size
    Most_Recent_Trade_Time
    Most_Recent_Trade_Market_Center
    Total_Volume
    Bid
    ...

>> data(2)
ans =
struct with fields:
    Symbol: 'GOOG'
    Most_Recent_Trade: 1078.99
    Most_Recent_Trade_Size: 1
    Most_Recent_Trade_Time: '19:58:47.052099'
    Most_Recent_Trade_Market_Center: 26
    Total_Volume: 0
    Bid: 1077.6
    Bid_Size: 100
    Ask: 1079.89
    Ask_Size: 200
    Open: []
    High: []
    Low: []
    Close: 1078.92
    Message_Contents: 'Cbav'
    Message_Description: 'Last qualified trade; A bid update
    occurred; An ask update occurred;
    A volume update occurred'
    Most_Recent_Trade_Conditions: '0517'
    Trade_Conditions_Description: 'Average Price Trade; Form-T Trade'
    Most_Recent_Market_Name: 'Direct Edge X (EDGX)'
```

If you have the Professional license of *IQC* and also Matlab's Parallel Computing Toolbox, then setting the **UseParallel** parameter to `true` (or 1) will process the quotes query for all the specified symbols in parallel (see discussion in §3.6). Note that in the case of quote queries, there is often little or no speedup in parallelization, because the delay is caused by waiting for market quote events, not due to CPU processing:

```
>> data = IQC('quotes', 'symbols',{'IBM','GOOG','AAPL'}, 'UseParallel',true);
```

Note that if you request quotes for a very large number of symbols in a single *IQC* command, and especially if you set **UseParallel** to `true`, you might run into your IQFeed account's symbols-limit (`MaxSymbols`; see §9.3). In such a case, IQFeed-generated error messages such as the following will be displayed on the Matlab console:

```
Warning: Requesting 3258 symbol quotes, which is more than your IQFeed account's
concurrent symbols limit (500) - quotes for some symbols may not be available.
(Type "warning off IQC:MaxSymbols" to suppress this warning.)
Level1 symbol limit reached - symbol 'IBM' not serviced!
```


By default, IQFeed reports 16 data fields for each quote: Symbol, Most Recent Trade, Most Recent Trade Size, Most Recent Trade Time, Most Recent Trade Market Center, Total Volume, Bid, Bid Size, Ask, Ask Size, Open, High, Low, Close, Message Contents, and Most Recent Trade Conditions.²⁰

If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary):

```
>> data = IQC('quotes', 'fields', {})
data =
    CurrentFields: {1x16 cell}
    AvailableFields: {1x68 cell}

>> data.AvailableFields
ans =
1x68 cell array
Columns 1 through 5
    {'Symbol'}    {'Exchange ID'}    {'Last'}    {'Change'}    {'Percent Change'}
Columns 6 through 11
    {'Total Volume'}    {'High'}    {'Low'}    {'Bid'}    {'Ask'}    {'Bid Size'}
Columns 12 through 17
    {'Ask Size'}    {'Tick'}    {'Range'}    {'Open Interest'}    {'Open'}    {'Close'}
Columns 18 through 22
    {'Spread'}    {'Settle'}    {'Delay'}    {'Restricted Code'}    {'Net Asset Value'}
...
```

If you have the Professional (or trial) *IQC* license, you can request IQFeed to report more than 50 additional data fields, and/or change the reported fields and their order, using the optional **Fields** parameter, as follows:

We can set **Fields** to 'All' (or 'all') to request all available data fields in reported quotes:²¹

```
>> data = IQC('quotes', 'Symbol', 'AAPL', 'Fields', 'all')
data =
    Symbol: 'AAPL'
    x7_Day_Yield: []
    Ask: 222.91
    Ask_Change: []
    Ask_Market_Center: 28
    Ask_Size: 100
    Ask_Time: '19:59:42.031900'
    Available_Regions: []
    Average_Maturity: []
    Bid: 222.71
    ...
```

The field names in the reported Matlab struct are the same as the IQField field names, except for the following minor changes to create valid Matlab field identifiers:

- spaces are replaced by '_' ("Ask Change" → Ask_Change)
- 'x' is prefixed to fields that start with numbers ("7 Day Yield" → x7_Day_Yield)

A complete table of available fields is provided for convenience at the bottom of this section. If you are uncertain about the meaning of a certain field, or wish to know

²⁰ The additional textual fields Message_Description, Trade_Conditions_Description and Most_Recent_Market_Name are IQC-generated textual interpretations of the codes in the IQFeed-generated Message_Contents, Trade_Conditions and Most_Recent_Trade_Market_Center fields, respectively, as governed by the **MsgParsingLevel** parameter (see §3.2).

²¹ Additional description fields will be generated by IQC for those fields that report value codes (for example, the Fraction Display Code and Financial Status Indicator fields), as governed by the **MsgParsingLevel** parameter (see §3.2).

which field reports certain data, please ask your DTN IQFeed representative (after all, *IQC* just reports the data as provided by IQFeed).

Some of the reported field values may be empty. For example, AAPL's `Average_Maturity` value is empty since this field is only relevant for bonds, not stocks. Similarly, EURUSD.FXCM's `Market_Capitalization` value is empty because Forex securities have no market cap. Likewise, `Net_Asset_Value` is only relevant for funds. `Delay=[]` indicates a real-time quote, whereas `Delay=15` indicates that the quote was delayed 15 minutes by the exchange (presumably because you do not possess a real-time data subscription for this exchange/security-type).²²

The **Fields** parameter can be set to any subset of `AvailableFields`,²³ as either a cell-array of strings, or as a comma-separated string. In this case, any subsequent quotes query will report the requested fields, in the specified order. For example:

```
>> data = IQC('quotes', 'Symbol','AAPL', 'Fields',{'Bid','Ask','Last'})
>> data = IQC('quotes', 'Symbol','AAPL', 'Fields','Bid,Ask,Last') %equivalent
data =
    Symbol: 'AAPL'
         Bid: 222.71
         Ask: 222.91
         Last: 222.11
```

The order of the specified **Fields** indicates the order in which the data fields will be reported. For example, to change the order of the reported data fields above:

```
>> data = IQC('quotes', 'Symbol','AAPL', 'Fields','Last,Ask,Bid')
data =
    Symbol: 'AAPL'
         Last: 222.11
         Ask: 222.91
         Bid: 222.71
```

Note that the `Symbol` field is always reported in the first position, regardless of whether or not it was specified in the **Fields** list, or of its specified position order in the **Fields** list (also note the optional spaces between the comma-separated field names):

```
>> data = IQC('quotes', 'Symbol','AAPL', 'Fields','Bid, Ask, Last, Symbol')
data =
    Symbol: 'AAPL'
         Bid: 222.71
         Ask: 222.91
         Last: 222.11
```

As noted, **Fields** can be set to any subset of the `AvailableFields`. If a bad field is specified (one which is not available in IQFeed), an error message will be displayed:

```
>> data = IQC('quotes', 'Symbol','AAPL', 'Fields','Bid, Ask, xyz')
Error using IQC
Bad field 'xyz' was requested in IQC quotes command (check the
capitalization/spelling).
Available fields are: 7 Day Yield, Ask, Ask Change, Ask Market Center, ...
```



Note: the more fields that you request, the longer the required processing time, by both IQFeed and *IQC*. To improve run-time performance and reduce latency, request only those data fields that are actually needed by your program.

²² See §9.3 for a programmatic method to determine whether your exchange subscription is delayed or real-time.

²³ `AvailableFields` is reported by an `IQC('quotes', 'fields', {})` command – see the previous page in this User Guide.

The following parameters affect quotes data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ²⁴	colon or comma-delimited string, or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> '@VX#' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).
NumOfEvents	integer	1	Returns up to the specified number of quotes
Timeout	number	5.0	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If set to true or 1, and the Parallel Toolbox is installed, querying multiple symbols is done in parallel (see §3.6; Pro <i>IQC</i> license only)
MaxWorkers	integer	(current parallel pool size)	Max number of parallel workers to use (up to the current pool size) when UseParallel =1
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Most Recent Trade, Most Recent Trade Size, Most Recent Trade Time, Most Recent Trade Market Center, Total Volume, Bid, Bid Size, Ask, Ask Size, Open, High, Low, Close, Message Contents, Most Recent Trade Conditions'	Sets the list of data fields reported by IQFeed for each quote. IQFeed's default set has 16 fields; 50+ additional fields can be specified (a detailed list of fields is provided below). If Fields is set to an empty value ({} or ''), the list of current, available fields is returned. If Fields is not empty, subsequent quotes queries will return the specified fields, in the specified order (Professional <i>IQC</i> license only). The Symbol field is always returned first, even if not specified. Examples: <ul style="list-style-type: none"> 'Bid:Ask:Last' 'Bid, Ask, Last' {'Bid', 'Ask', 'Last'} 'All' (indicates all available fields)

The full list of available fields in IQFeed as of July 2019 is listed below. Note that some of these fields may not be available, and IQFeed may also add/modify this list at any time. The list of fields that are actually available can be retrieved in *IQC* using the `IQC('quotes', 'fields', {})` command, as explained above. For details about any of these fields, please contact your DTN/IQFeed representative (*IQC* just reports the data, it has no control over the reported values or definition of the data fields).

²⁴ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Field Name	Field Type	Description	Data origin ²⁵
7 Day Yield	float	Value of a Money Market fund over past 7 days.	Exchange/other
Ask	float	Min price a market maker/broker accepts for a security.	Exchange/other
Ask Change	float	Change in Ask since last offer.	IQConnect
Ask Market Center	integer	Market Center that sent the ask information (see §8.3).	Exchange/other
Ask Size	integer	The share size available for the ask price	Exchange/other
Ask Time	hh:mm:ss.ffffff	The time of the last ask. May be reported as 99:99:99 outside trading hours to indicate an invalidated quote. ²⁶	Exchange/other
Available Regions	string	Dash-delimited list of available regional exchanges.	IQConnect
Average Maturity	float	Average number of days until maturity of a Money Market Fund's assets.	Exchange/other
Bid	float	Max price a market maker/broker will pay for a security.	Exchange/other
Bid Change	float	Change in Bid since last offer.	IQConnect
Bid Market Center	integer	Market Center that sent the bid information (see §8.3).	Exchange/other
Bid Size	integer	The share size available for the bid price.	Exchange/other
Bid Time	hh:mm:ss.ffffff	The time of the last bid. May be reported as 99:99:99 outside trading hours to indicate an invalidated quote. ²⁷	Exchange/other
Change	float	Today's change (Last - Close)	IQConnect
Change From Open	float	Change in last price since last open.	IQConnect
Close	float	The closing price of the day. For commodities this will be the last trade price of the session.	Exchange/other
Close Range 1	float	For commodities only. Range value for closing trades that aren't reported individually.	Exchange/other
Close Range 2	float	For commodities only. Range value for closing trades that aren't reported individually.	Exchange/other
Days to Expiration	string	Number of days to contract expiration.	IQConnect
Decimal Precision	string	Last Precision used.	DTN
Delay	integer	The number of minutes a quote is delayed when not authorized for real-time data. ²⁸	Exchange/other
Exchange ID	hexadecimal	The Exchange Group ID.	DTN
Extended Trade	float	Price of the most recent extended trade (last qualified trades + Form T trades).	Exchange/other
Extended Trade Date	MM/DD/CCYY	Date of the extended trade.	Exchange/other
Extended Trade Market Center	integer	Market Center of the most recent extended trade (last qualified trades + Form T trades); see §8.3.	Exchange/other
Extended Trade Size	integer	Size of the most recent extended trade (last qualified trades + Form T trades).	Exchange/other
Extended Trade Time	hh:mm:ss.ffffff	Time (including microseconds) of the most recent extended trade (last qualified trades + Form T trades).	Exchange/other
Extended Trading Change	float	Extended Trade minus Yesterday's close.	IQConnect
Extended Trading Difference	float	Extended Trade minus Last.	IQConnect

²⁵ In this table, "exchange/other" means either the exchange, or some other 3-party that provides data to DTN/IQFeed.

²⁶ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=3891>

²⁷ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=3891>

²⁸ See §9.3 for a programmatic method to determine whether your exchange subscription is delayed or real-time.

Field Name	Field Type	Description	Data origin ²⁵
Financial Status Indicator	char	Denotes if an issuer has failed to submit its regulatory filings on a timely basis, has failed to meet the exchange's continuing listing standards and/or filed for bankruptcy. A corresponding description field will be generated by <i>IQC</i> for this field when MsgParsingLevel ≥ 2 (see §3.2)	Exchange/other
Fraction Display Code	string	Display formatting code. A corresponding description field will be generated by <i>IQC</i> for this field when MsgParsingLevel ≥ 2 (see §3.2)	DTN
High	float	Today's highest trade price.	Exchange/other
Last	float	Last trade price from the regular trading session.	Exchange/other
Last Date	MM/DD/CCYY	Date of the last qualified trade.	Exchange/other
Last Market Center	integer	Market Center of most recent last qualified trade.	Exchange/other
Last Size	integer	Size of the most recent last qualified trade.	Exchange/other
Last Time	hh:mm:ss.ffffff	Time (including microseconds) of the most recent last qualified trade.	Exchange/other
Low	float	Today's lowest trade price.	Exchange/other
Market Capitalization	float	Real-time calculated market cap (Last price * Common Shares Outstanding).	IQConnect
Market Open	integer	1 = market open, 0 = market closed. Note: valid for Futures and Future Options only.	DTN
Message Contents	non-delimited string of single characters of message identification code	<p>Possible single character values include:</p> <ul style="list-style-type: none"> C – Last Qualified Trade E – Extended Trade = Form T trade O – Other Trade = Any trade not accounted for by C or E b – A bid update occurred a – An ask update occurred o – An Open occurred h – A High occurred l – A Low occurred c – A Close occurred s – A Settlement occurred v – A volume update occurred <p>Notes: you can get multiple codes in a single message but you will only get one trade identifier per message. It is also possible to receive no codes in a message if the fields that updated were not trade or quote related.</p> <p>A corresponding description field is generated by <i>IQC</i> for this field when MsgParsingLevel ≥ 2 (see §3.2)</p>	IQConnect
Most Recent Trade	float	Price of most recent trade (inc. non-last-qualified trades).	Exchange/other
Most Recent Trade Aggressor	integer	<p>Identifies if the trade was executed at the bid or the ask price. As of IQFeed 6.1, this feature is only supported for CME and ICE exchanges; unsupported symbols return [].</p> <p>Possible values:</p> <ul style="list-style-type: none"> 0 – invalid or unsupported 1 – most recent trade was executed at the Ask price 2 – most recent trade was executed at the Bid price 3 – most recent trade was executed at some other price [] – unknown or unsupported <p>(requires IQFeed client 6.1 or newer)</p>	<p>Exchange/other</p> <p>(only provided by some exchanges)</p>
Most Recent Trade Conditions	string of 2-digit hex numbers	Conditions that identify the type of most recent trade. A corresponding description field is generated by <i>IQC</i> for	Exchange/other

Field Name	Field Type	Description	Data origin ²⁵
		this field when MsgParsingLevel ≥ 2 (see §3.2, §8.7)	
Most Recent Trade Date	MM/DD/CCYY	Date of most recent trade.	Exchange/other
Most Recent Trade Day Code	integer	The day of month (1-31) in which the most recent trade occurred. (requires IQFeed client 6.1 or newer)	Exchange/other
Most Recent Trade Market Center	integer	Market Center of most recent trade. A corresponding description field will be generated by <i>IQC</i> for this field when MsgParsingLevel ≥ 2 (see §3.2, §8.3)	Exchange/other
Most Recent Trade Size	integer	Size of most recent trade.	Exchange/other
Most Recent Trade Time	hh:mm:ss.ffffff	Time (including microseconds) of most recent trade.	Exchange/other
Net Asset Value	float	The market value of a mutual fund share. Equal to net assets / total number of shares outstanding. Duplicates the Bid field. Valid for Mutual Funds only.	Exchange/other
Number of Trades Today	integer	The number of trades for the current day.	IQConnect/DTN
Open	float	The opening price of the day. For commodities this will be the first trade of the session.	Exchange/other
Open Interest	integer	IEOptions, Futures, FutureOptions, SSFutures only.	Exchange/other
Open Range 1	float	For commodities only. Range value for opening trades that aren't reported individually.	Exchange/other
Open Range 2	float	For commodities only. Range value for opening trades that aren't reported individually.	Exchange/other
Percent Change	float	= Change / Close	IQConnect
Percent Off Average Volume	float	Current Total Volume / Average Volume	IQConnect
Previous Day Volume	integer	Previous Day's Volume.	Exchange/other
Price-Earnings Ratio	float	Real-time calculated PE (Last / Earnings Per Share).	IQConnect
Range	float	Trading range for the current day (high - low)	IQConnect
Restricted Code	string	"N"=Short Sale is not restricted, "R"=Restricted.	Exchange/other
Settle	float	Settle price (Futures or FutureOptions only).	Exchange/other
Settlement Date	MM/DD/YYYY	The date that the Settle field is valid for.	Exchange/other
Spread	float	The difference between Bid and Ask prices.	IQConnect
Symbol	string	The symbol name of the security	IQConnect
Tick	integer	173=Up, 175=Down, 183=No Change. Based on the previous trade. Only valid for Last qualified trades.	IQConnect
TickID	integer	Identifier for tick (not necessarily sequential).	DTN
Total Volume	integer	Today's cumulative volume in number of shares.	IQConnect,DTN or exchange
Type	string	"Q"=Update message, "P"=Summary Message.	IQConnect
Volatility	float	Real-time calculated volatility: (High - Low) / Last.	IQConnect
VWAP	float	Volume Weighted Average Price.	IQConnect/DTN

4.2 Fundamental information

Fundamental data on a symbol can be fetched using a 'fundamental' action, as follows:

```
>> data = IQC('fundamental', 'symbol', 'IBM')
data =
      Symbol: 'IBM'
      Exchange_ID: 7
      PE: 25.7
      Average_Volume: 4588000
      x52_Week_High: 180.95
      x52_Week_Low: 139.13
      Calendar_Year_High: 171.13
      Calendar_Year_Low: 144.395
      Dividend_Yield: 3.79
      Dividend_Amount: 1.5
      Dividend_Rate: 6
      Pay_Date: '03/10/2018'
      Ex_dividend_Date: '02/08/2018'
      Short_Interest: 17484332
      Current_Year_EPS: 6.17
      Next_Year_EPS: []
      Five_year_Growth_Percentage: -0.16
      Fiscal_Year_End: 12
      Company_Name: 'INTERNATIONAL BUSINESS MACHINE'
      Root_Option_Symbol: 'IBM'
      Percent_Held_By_Institutions: 59.9
      Beta: 1.05
      Leaps: []
      Current_Assets: 49735
      Current_Liabilities: 37363
      Balance_Sheet_Date: '12/31/2017'
      Long_term_Debt: 39837
      Common_Shares_Outstanding: 921168
      Split_Factor_1: '0.50 05/27/1999'
      Split_Factor_2: '0.50 05/28/1997'
      Market_Center: []
      Format_Code: 14
      Precision: 4
      SIC: 7373
      Historical_Volatility: 25.79
      Security_Type: 1
      Listed_Market: 7
      x52_Week_High_Date: '03/08/2017'
      x52_Week_Low_Date: '08/21/2017'
      Calendar_Year_High_Date: '01/18/2018'
      Calendar_Year_Low_Date: '02/09/2018'
      Year_End_Close: 153.42
      Maturity_Date: []
      Coupon_Rate: []
      Expiration_Date: []
      Strike_Price: []
      NAICS: 541512
      Exchange_Root: []
      Option_Premium_Multiplier: []
      Option_Multiple_Deliverable: []
      Session_Open_Time: []
      Session_Close_Time: []
      Base_Currency: []
      Contract_Size: []
      Contract_Months: []
      Minimum_Tick_Size: []
      First_Delivery_Date: []
      FIGI: 'BBG000BLNNH6'
```



```

Security_SubType: []
Price_Format_Description: 'Four decimal places'
Exchange_Description: 'New York Stock Exchange (NYSE)'
Listed_Market_Description: 'New York Stock Exchange (NYSE)'
Security_Type_Description: 'Equity'
Security_SubType_Description: ''
SIC_Description: 'COMPUTER INTEGRATED SYSTEMS DESIGN'
NAICS_Description: 'Computer Systems Design Services'
SIC_Sector_Name: 'Services'
NAICS_Sector_Name: 'Professional, Scientific, Technical services'

```

Notes:

1. the naming, interpretation and order of returned data fields is controlled by IQFeed, not by IQC – DTN might change these fields in the future.
2. Splits – when only one split is available, Split_Factor_2 will be empty; when no splits are known to IQFeed, both split fields will be empty. Splits are reported as '#.## <mm/dd/yyyy date>', i.e. an American-format date rounded to 2 decimal digits.²⁹ In any case, only the last 2 splits are reported by IQFeed.³⁰
3. the inclusion of the *_Description fields (Price_Format_Description, Exchange_Description, etc.) depends on the **MsgParsingLevel** parameter having value of 2 or higher (see §3.2 for details). When **MsgParsingLevel** is 1 or 0, these fields will not be part of the returned data struct.
4. Depending on your IQFeed client version, additional fundamental data fields may be returned. For example, the following fields were added in client 6.1 (some fields only have values for certain security types, e.g. futures/options): Session_Open_Time, Session_Close_Time, Base_Currency, Contract_Size, FIGI, Contract_Months, Minimum_Tick_Size, First_Delivery_Date, and Security_SubType. For example, (redacted for clarity):

```

>> data = IQC('fundamental', 'symbol', '@ES#')
data =
  struct with fields:
      Symbol: '@ES#'
      Exchange_ID: 34
      PE: []
      Average_Volume: []
      x52_Week_High: 3006
      x52_Week_Low: 2350
      ...
      Company_Name: 'E-MINI S&P 500 SEPTEMBER 2019'
      Expiration_Date: '09/20/2019'
      Exchange_Root: 'ES'
      Session_Open_Time: '18:00:00'
      Session_Close_Time: '17:00:00'
      Base_Currency: 'USD'
      Contract_Size: 50
      Contract_Months: '--H--M--U--Z'
      Minimum_Tick_Size: 0.25
      Underlying_Contract: '@ESU19'
      Exchange_Description: 'Chicago Mercantile Exchange (CME)'
      Listed_Market_Description: 'Chicago Mercantile Exchange Mini Sized
      Contracts (CMEMINI)'
      Security_Type_Description: 'Future'

```

²⁹ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=5582>. Note that this could lead to numeric inaccuracies, for example GOOGL's split on 4/3/2014 is reported as 0.50 rather than the more accurate $1:1.998 = 0.5005$. Also note that some splits (e.g. GOOGL's 1:2.002 split on 3/27/2014) are not reported by IQFeed for some reason (probably a data error).

³⁰ For additional (and more accurate) splits history, refer to <https://www.stocksplithistory.com>

To improve run-time performance, fundamental data is cached in Matlab memory for each symbol. IQFeed is queried for fundamental data for a symbol only if the symbol was not previously queried in the current Matlab session. This is typically a good thing, because fundamental data is relatively stable. To force *IQC* to re-query IQFeed for fundamental data even when it is cached, set the **Debug** parameter:

```
>> data = IQC('fundamental', 'symbol', 'AAPL', 'debug', true);
>> data = IQC('debug', false); % exit debug mode
```

It is possible to fetch fundamental data of multiple symbols in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols: ³¹

```
>> data = IQC('fundamental', 'symbols', 'AAPL:GOOG') %or: {'AAPL', 'GOOG'}
data =
    1x2 struct array with fields:
        Symbol
        Exchange_ID
        PE
        ...

>> data(1)
ans =

        Symbol: 'AAPL'
    Exchange_ID: 5
           PE: 20.4
Average Volume: 26900000
    x52_Week_High: 228.87
    x52_Week_Low: 149.16
           ...

>> data(2)
ans =

        Symbol: 'GOOG'
    Exchange_ID: 5
           PE: 51.9
Average Volume: 1239000
    x52_Week_High: 1273.89
    x52_Week_Low: 909.7
Calendar_Year_High: 1273.89
           ...
```

The list of fundamental data fields in IQFeed as of 1/7/2019 is listed below. Note that IQFeed may modify this list at any time (for example, IQFeed client 6.1 has added several fields). For details about any of these fields, please ask DTN/IQFeed. *IQC* just reports the data, it has no control over the reported values or definition of data fields. Note that the fundamental data fields cannot be modified, unlike quotes fields.

Field Name	Field Type	Description	Data origin ³²
Symbol	string	The Symbol ID to match with watch request	DTN
Exchange ID	hexadecimal	This is the Exchange Group ID (code). Convert to decimal and use the Listed Markets lookup to decode this value.	DTN
PE	float	Price/Earnings ratio	Exchange/other
Average Volume	integer	Average daily volume (4 week average)	DTN
52 Week High	float	Highest price of the last 52 weeks. For futures, this is the contract High.	DTN

³¹ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

³² In this table, “exchange/other” means either the exchange, or some other 3-party that provides data to DTN/IQFeed.

Field Name	Field Type	Description	Data origin ³²
52 Week Low	float	Lowest price of the last 52 weeks. For futures, this is the contract Low.	DTN
Calendar year high	float	High price for the current calendar year.	DTN
Calendar year low	float	Low price for the current calendar year.	DTN
Dividend yield	float	The annual dividends per share paid by the company divided by current market price per share of stock, as a % value.	Exchange/other
Dividend amount	float	The current quarter actual dividend	Exchange/other
Dividend rate	float	The annualized dividend expected to be paid by company	Exchange/other
Pay date	MM/DD/YYYY	Date on which a company made its last dividend payment	Exchange/other
Ex-dividend date	MM/DD/YYYY	The actual date in which a stock goes ex-dividend, typically about 3 weeks before the dividend is paid to shareholders of record. The amount of the dividend is reflected in a reduction of the share price on this date.	Exchange/other
Short Interest	integer	Total number of shares that were sold short and were not repurchased to settle outstanding short market positions. Valid data is only available for some exchanges/sec-types. ³³	3rd party
Current year earnings per share	float	The portion of a company's profit allocated to each outstanding share of common stock	Exchange/other
Next year earnings per share	float	Total amount of earnings per share a company is estimated to accumulate over the next 4 quarters of current fiscal year	Exchange/other
Five-year growth percentage	float	Earnings Per Share growth rate over a five year period	Exchange/other
Fiscal year end	integer	The numeric month in which the company's fiscal year ends. For example, 4=April, 10=October.	Exchange/other
Company name	string	Company name or contract description	DTN
Root Option symbol	string	A space separated list (there may be more than one)	Exchange/other
Percent held by institutions	float	A percentage of outstanding shares held by banks and institutions.	Exchange/other
Beta	float	A coefficient measuring a security's relative volatility: the covariance of this security's price in relation to the rest of the market. 30 day historical volatility.	Exchange/other
Leaps (there may be more than one)	string	Long term equity anticipation securities	Exchange/other
Current assets	float	The amount of total current assets held by a company as of a specific date in Millions (lastADate)	Exchange/other
Current liabilities	float	The amount of total current liabilities held by a company as of a specific date in Millions (lastADate).	Exchange/other
Balance sheet date	MM/DD/YYYY	Last date that a company issued their quarterly report.	Exchange/other
Long-term debt	float	The amount of long term debt held by a company as of a specific date in Millions (lastADate).	Exchange/other
Common shares outstanding	float	The amount of common shares outstanding (in thousands).	Exchange/other
Split factor 1	string	A float a space, then MM/DD/YYYY. For example: '0.5 12/20/2013'	Exchange/other
Split factor 2	string	A float a space, then MM/DD/YYYY. For example: '0.5 12/20/2013'	Exchange/other

³³ <http://forums.dtn.com/index.cfm?page=topic&topicID=5787>

Field Name	Field Type	Description	Data origin ³²
Format Code	string	Display format code	DTN
Precision	integer	Number of decimal digits	DTN
SIC	integer	Standard Industrial Classification – a 4-digit federal code that identifies the security's specific industry.	Exchange/other
Historical Volatility	float	30-trading day volatility, calculated using Black-Scholes	DTN
Security Type	string	The security type code	DTN
Listed Market	string	The listing market ID	DTN
52 Week High Date	MM/DD/YYYY	The date of the highest price of the last 52 weeks. For futures, this is the contract High Date.	DTN
52 Week Low Date	MM/DD/YYYY	The date of the lowest price of the last 52 weeks. For futures, this is the contract Low Date.	DTN
Calendar Year High Date	MM/DD/YYYY	Date at which the High price for the current calendar year occurred.	DTN
Calendar Year Low Date	MM/DD/YYYY	Date at which the Low price for the current calendar year occurred.	DTN
Year End Close	float	Price of Year End Close (Equities Only)	DTN
Maturity Date	MM/DD/YYYY	Date of maturity for a Bond.	DTN
Coupon Rate	float	Interest Rate for a Bond.	Exchange/other
Expiration Date	MM/DD/YYYY	IEOptions, Futures, FutureOptions, and SSFutures only	IEOptions by the exchange; others by DTN
Strike Price	float	IEOptions only	Exchange/other
NAICS	integer	North American Industry Classification System code ³⁴	3rd party
Exchange Root	string	The root symbol under which this symbol is listed at the exchange.	Exchange
Options Premium Multiplier	float	IEOptions only	3rd party
Options Multiple Deliverables	integer	IEOptions only. 1 means they exist, 0 means they do not.	3rd party
Session Open Time	hh:mm:ss	Futures and Future Options Only (IQFeed 6.1 or newer)	3rd party
Session Close Time	hh:mm:ss	Futures and Future Options Only (IQFeed 6.1 or newer)	3rd party
Base Currency	string	Futures and Future Options Only (IQFeed 6.1 or newer)	3rd party
Contract Size	string	Deliverable quantity of a future or option contract. (IQFeed 6.1 or newer)	3rd party
Contract Months	string	non-delimited string of upper-case single character month codes (IQFeed 6.1 or newer)	3rd party
Minimum Tick Size	float	Minimum price movement. (IQFeed 6.1 or newer)	3rd party
First Delivery Date	MM/DD/YYYY	Futures and Future Options Only (IQFeed 6.1 or newer)	3rd party
FIGI	string	Financial Instrument Global Identifier ³⁵ (IQFeed 6.1 or newer). For example: 'BBG000BLNNH6' (IBM)	3rd party
Security SubType	integer	The security's SubType code (IQFeed 6.1 or newer). 1=Binary option, 2=weekly option, 3=ETF, []=other	DTN
UnderlyingContract	string	The specific future contract that underlies a continuous future. For example, for @ES# this might be '@ESU19' ³⁶	IQC

³⁴ <https://www.census.gov/eos/www/naics>

³⁵ <https://openfigi.com>, <https://omg.org/figi>

4.3 Interval bars

Interval bars data for one or more symbols can be fetched using the 'intervalbars' action. For example, to fetch the latest 60-second interval bar for the current E-Mini contract:

```
>> data = IQC('intervalbars', 'Symbol','@ES#')
data =
      Symbol: '@ES#'
      BarType: 'Complete interval bar from history'
      Timestamp: '2018-09-05 12:57:00'
      Open: 2887.75
      High: 2888.25
      Low: 2887.5
      Close: 2888.25
      CummlativeVolume: 1117565
      IntervalVolume: 913
      NumberOfTrades: 0
```

In the returned data struct, we can see the following fields:

- **Symbol** – the requested Symbol.
- **BarType** – typically ‘Complete interval bar from history’, but in some cases might be ‘Complete interval bar from stream’ or ‘Updated interval bar’.
- **Timestamp** – server timestamp (string format) for this interval bar. The timestamp is of the end of the bar, not the beginning.
- **Open** – price at the start of this interval bar.
- **High** – highest price during this interval bar.
- **Low** – lowest price during this interval bar.
- **Close** – price at the end of this interval bar.
- **CummlativeVolume** – total trade volume since start of the current trading day.
- **IntervalVolume** – trade volume during this interval bar.
- **NumberOfTrades** – number of trades during this interval bar. Relevant only when **IntervalType** is set to 'ticks'/'trades'.

The **IntervalType** (default: 'secs') and **IntervalSize** (default: 60) parameters should typically be specified together. Note that **IntervalSize** must be a positive integer value (i.e. its value cannot be 4.5 or 0). If **IntervalType** is 'ticks'/'trades', **IntervalSize** must be 2 or higher. If **IntervalType** is 'volume', **IntervalSize** must be 100 or higher. If **IntervalType** is 'secs', **IntervalSize** must be any integer between 1-300 (5 minutes), or any multiple of 60 (1 minute) between 300-3600 (1 hour), or 7200 (2 hours).³⁷

We can ask for multiple bars by setting **NumOfEvents** or **MaxItems** to a positive integer, resulting in an array of structs in the format above (empty array if no data is available):

```
>> data = IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',4)
data =
      4x1 struct array with fields:
```

³⁶ Unlike all other fields, this field is not reported by IQFeed but rather computed by IQC, based on the reported fields (Symbol and Expiration Date). It contains a non-empty value only for continuous future contracts (e.g. @ES#).

³⁷ Note that IQFeed’s limitations on live 'secs' interval bars are stricter than the limitations on historical interval bars (§5.4): <http://forums.dtn.com/index.cfm?page=topic&topicID=5529>

```

Symbol
BarType
...
>> data(1)
ans =
        Symbol: '@VX#'
        BarType: 'Complete interval bar from history'
        Timestamp: '2018-09-05 12:36:00'
            Open: 14.45
            High: 14.5
            Low: 14.45
            Close: 14.45
CummlativeVolume: 57077
IntervalVolume: 17
NumberOfTrades: 0

```

IQFeed only returns interval bars that had market ‘action’. Other bars are not sent from IQFeed – they will appear in *IQC*’s returned data as gaps in the `Timestamp`.

Also note that it is possible that not all the requested bars will be received before *IQC*’s timeout (default value: 5 secs) returns the results:

```

>> data = IQC('intervalbars', 'Symbol','IBM', 'NumOfEvents',4)

Warning: IQC timeout: only partial data is returned. Perhaps the Timeout
parameter should be set to a value larger than 5 or the NumOfEvents parameter
to a value smaller than 4

data =
2x1 struct array with fields:
    Symbol
    BarType
    ...

```

To control the maximal duration that *IQC* will wait for the data, set the **Timeout** parameter. For example, to wait up to 60 secs to collect 4 bars:

```

>> data = IQC('intervalbars', 'Symbol','IBM', 'NumOfEvents',4, 'timeout',60);

```

Interval bars query fetches historical bars data, starting from the date/time that is set by the **BeginDateTime** parameter (see the parameters table below). This is similar to (and subject to the same limitations as) fetching historical interval data (see §5.4), but with no specified end point. *IQC* will return both the historical bars, as well as new (live) real-time streaming interval bars, as they become available. **BeginDateTime**’s default value is 00:00:00 today (server time), so you will almost always get historical bars before live streaming bars. If you run the query at mid-day, you may get hundreds of historical bars before you get the first live streaming bar. So, if you set **NumOfEvents** to a low value, you might receive only historical bars, without any live streaming bars.

Unlike quotes (§4.1), when you specify **NumOfEvents** > 1, *IQC* does not wait for new bars to arrive; instead, it returns previous (historic) bars, as long as this does not conflict with the specified **BeginDateTime**. For example, if you set **NumOfEvents**=5, you will receive the latest 5 bars: 4 complete historic bars, as well as the current (incomplete) bar. If you require live (future) interval bars, then set **BeginDateTime**, or use the streaming mechanism that is described in §6.3. For example, if you set **BeginDateTime** to 5 bars ago and **NumOfEvents**=15, then *IQFeed* will return the 5 historic bars and wait for 10 additional future bars (subject to the specified **Timeout**).

Additional data filtering parameters: **MaxDays**, **BeginFilterTime** and **EndFilterTime**.

You can query multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('intervalbars', 'symbols', {'IBM','GOOG','AAPL'});
>> data = IQC('intervalbars', 'symbols', 'IBM:GOOG:AAPL'); % equivalent
```

If the query returns the same number of data elements for all symbols, the results will be returned as a struct array, with columns corresponding to the requested symbols:

```
data =
100x3 struct array with fields:
    Symbol
    BarType
    Timestamp
    Open
    High
    Low
    Close
    CumulativeVolume
    IntervalVolume
    NumberOfTrades
```

However, if *IQC* returns a different amount of data for various symbols, the results are returned as a cell array, with cell elements corresponding to the requested symbols. For example, in the following query, there is no symbol 'XXX' so *IQC* returns empty results for this particular symbol:³⁸

```
>> data = IQC('intervalbars', 'Symbol', 'IBM:GOOG:XXX', 'UseParallel', true)
data =
1x3 cell array
    {100x1 struct}    {100x1 struct}    {0x0 double}
```

If you have the Professional license of *IQC* and also Matlab's Parallel Computing Toolbox, then setting the **UseParallel** parameter to `true` (or 1) will process the quotes query for all the specified symbols in parallel (see discussion in §3.6):

```
>> data = IQC('intervalbars', 'symbols', {'IBM','GOOG','AAPL'}, ...
    'UseParallel', 1);
```

The following parameters affect interval bars data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ³⁹	colon or comma-delimited string or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: <ul style="list-style-type: none"> '@VX#' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0. Multiple symbols can be parallelized using the UseParallel parameter (see below).

³⁸ The **UseParallel** parameter is set here in order to avoid the run-time error of "Symbol 'XXX' was not found"

³⁹ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols will be done in parallel (see §3.6; Professional <i>IQC</i> license only).
MaxWorkers	integer	(the current parallel pool size)	Maximal number of parallel workers to use (up to the current pool size) when UseParallel =true
NumOfEvents	integer	Inf	One of: <ul style="list-style-type: none"> • inf – continuous endless streaming interval bars for specified symbol(s) • N>1 – stream only N interval bars • 1 – get only a single interval bar • 0 – stop streaming interval bars • -1 – return latest interval bars data while continuing to stream new bars
MaxItems	integer	100	Returns up to the specified number of bars (if available).
MaxDays	integer	1	Max number of trading days to retrieve
IntervalType	string	'secs'	Sets the type of interval size. One of the following values: <ul style="list-style-type: none"> • 's' or 'secs' – time [seconds] (default) • 'v' or 'volume' – traded volume • 't' or 'ticks' – number of ticks
IntervalSize	integer	60	Size of bars in IntervalType units. Must be ≥ 1 for secs, ≥ 2 for ticks, ≥ 100 for volume.
BeginFilterTime	string	'00:00:00'	Only return bars that begin after this time of day (US Eastern time-zone). Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
EndFilterTime	string	'23:59:59'	Only return bars that end before this time of day (US Eastern time-zone). Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
BeginDateTime	integer or string or datetime object	" (empty string) meaning today at 00:00:00	Only return bars that begin after this date/time (US Eastern time-zone). Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc.
Timeout	number	5.0	Max number of seconds to wait (0-9000) for data in blocking mode (0 means infinite)

4.4 Market depth (Level 2)

Level 2 (II) market data on a symbol can be fetched using a 'marketdepth' action. Deep order-book rows (market *depth*) are reported for futures, and the top Bid/Ask for each separate market maker (market *width*) are reported for equities.⁴⁰ The following example fetches market-depth data for the S&P E-Mini continuous future:

```
>> data = IQC('marketdepth', 'symbol', '@ES#')
data =
    10x1 struct array with fields:
        Symbol
        ID
        Bid
        Ask
        BidSize
        AskSize
        BidTime
        Date
        AskTime
        BidInfoValid
        AskInfoValid
        Condition
        Condition_Description
        ID_Description
```

The latest data (i.e., state of the market-depth table) is returned as a Matlab struct array, whose elements correspond to the market-depth rows. For example, to see the data for row #3 (i.e., 2 rows below the top-of-book row), you can access array element #3:

```
>> data(3)
ans =
    struct with fields:
        Symbol: '@ES#'
        ID: 'MD03'
        Bid: 2926.75
        Ask: 2928
        BidSize: 94
        AskSize: 129
        BidTime: '03:21:15.037291'
        Date: '2019-05-02'
        AskTime: '03:21:19.064197'
        BidInfoValid: 1
        AskInfoValid: 1
        Condition: 52
        Condition_Description: 'regular'
        ID_Description: 'Order book row #3'
```

`BidInfoValid` and `AskInfoValid` values are logical (true/false) values, which appear as 1 or 0 (respectively) in the returned data struct. The `ID` field ('MD03' in this case) indicates that this is market-depth row 3 (also note the related `ID_Description` field).

Each incoming market depth message provides information on a single Level 2 data row. IQFeed's messages arrive at a random, unsorted, unpredictable order in two groups: first the messages that define the current (snapshot) baseline of all rows, then update messages for individual rows when traders add, cancel or modify orders.

⁴⁰ Until IQFeed version 6.1, IQFeed did not offer deep market depth for equities, nor market width for futures. For such detailed information you could use premium services such as Nasdaq TotalView (<http://nasdaqtrader.com/Trader.aspx?id=Totalview2>). IQFeed 6.2 enabled deep market depth (so-called "Market-By-Order"), which are supported starting with IQC version 2.30.

Depending on the requested **Symbol**, IQFeed may report 5 to 15 market-depth rows. To ensure that all the baseline data rows are received, set the **NumOfEvents** parameter to at least the total number of rows expected for the **Symbol**. For example:

```
>> data = IQC('marketdepth', 'symbol', '@ES#', 'NumOfEvents', 15)
data =
    10x1 struct array with fields:
        Symbol
        ID
        Bid
        Ask
        BidSize
        AskSize
        ...
```

The returned struct array can be converted into a Matlab table object, as follows:⁴¹

```
>> struct2table(data)
ans =
    10x14 table
        Symbol      ID      Bid      Ask      BidSize  AskSize      BidTime ...
    _____
    '@ES#'      'MD01'    2927.25    2927.5      58      121    '03:21:25.213504'
    '@ES#'      'MD02'      2927    2927.75      78      111    '03:21:22.040253'
    '@ES#'      'MD03'    2926.75      2928      94      129    '03:21:15.037291'
    '@ES#'      'MD04'    2926.5    2928.25      95      107    '03:21:04.023779'
    '@ES#'      'MD05'    2926.5    2928.75     104      184    '03:42:19.025285'
    '@ES#'      'MD06'    2926.25      2929     123      181    '03:42:43.020801'
    '@ES#'      'MD07'      2926    2929.25     137      127    '03:43:01.042949'
    '@ES#'      'MD08'    2925.75    2929.5       86      135    '03:42:01.029094'
    '@ES#'      'MD09'    2925.5    2929.75     183      161    '03:42:02.021818'
    '@ES#'      'MD10'    2925.25      2930     152      382    '03:42:07.003202'
```

For equities, IQFeed returns market *width* (not *depth*) data – the top bid/ask row (but not order-book depth) for each market maker that trades the equity. For example:

```
>> data = IQC('marketdepth', 'symbol', 'MSFT', 'NumOfEvents', 50)
data =
    4x1 struct array with fields:
        Symbol
        ID
        Bid
        Ask
        ...

>> data(1)
ans =
    struct with fields:
        Symbol: 'MSFT'
        ID: 'BATS'
        Bid: 0
        Ask: 129.39
        BidSize: 0
        AskSize: 600
        BidTime: '18:29:49.000347'
        Date: '2019-05-01'
        AskTime: '99:99:99.000000'
        BidInfoValid: 0
        AskInfoValid: 1
        Condition: 52
        Condition_Description: 'regular'
        ID_Description: 'CBOE TRADING, INC.'
```

⁴¹ Some fields at the table's right side are not shown here due to space limitations, but are available in the actual Matlab object

```
>> data(2)
ans =
  struct with fields:
      Symbol: 'MSFT'
      ID: 'NSDQ'
      Bid: 127.91
      Ask: 130.09
      BidSize: 100
      AskSize: 100
      BidTime: '04:03:41.004392'
      Date: '2019-05-02'
      AskTime: '04:17:07.020285'
      BidInfoValid: 1
      AskInfoValid: 1
      Condition: 52
      Condition_Description: 'regular'
      ID_Description: 'Nasdaq Execution Services'

>> struct2table(data2)
ans =
  4x14 table
      Symbol    ID      Bid      Ask      BidSize AskSize      BidTime      Date ...
      _____
      'MSFT'    'BATS'      0      129.39      0      600      '18:29:49.000347' '2019-05-01'
      'MSFT'    'NSDQ'    127.91    130.09     100     100      '04:03:41.004392' '2019-05-02'
      'MSFT'    'EDGX'    127.54    127.98     200    4800      '19:53:35.049950' '2019-05-01'
      'MSFT'    'ARCX'    127.97    128.71     100     100      '04:17:13.037004' '2019-05-02'
```

In this example, which was ran outside regular trading hours (early morning of 2019-05-02), BATS reported a valid Ask from the previous evening, but no valid Bid. The other market makers (NSDQ, EDGX and ARCX) reported both valid Bid and Ask.

Note that the reported market-maker rows are not sorted. Depending on the time of your query, you may receive a different set of market-makers: market makers who do not have any valid Bid/Ask are not reported.⁴³ Depending on the specific equity and request time, you may receive between 0 and dozens of market maker rows.

Using the reported Level 2 data from different market makers enables arbitrage trading: buying a security from market maker A (who offers the lowest Ask price) and selling to market maker B (who offers the highest Bid price).

If you wish to receive Level 2 quotes even when they are empty (invalid Bid and Ask), set the **IncludeEmptyQuotes** parameter to true or 1 (default value: false). For example:

```
>> struct2table(IQC('marketdepth', 'symbol', 'MSFT', 'NumOfEvents', 70, ...
      'IncludeEmptyQuotes', true))
ans =
  67x14 table
      Symbol    ID      Bid      Ask      BidSize AskSize      BidTime      Date ...
      _____
      'MSFT'    'SDLR'      0      0      0      0      '99:99:99.000000' '2019-05-01'
      'MSFT'    'STXG'      0      0      0      0      '99:99:99.000000' '2019-05-01'
      'MSFT'    'BATS'      0    129.39      0     600      '18:29:49.000347' '2019-05-01'
      'MSFT'    'NSDQ'    127.91    130.09     100     100      '04:03:41.004392' '2019-05-02'
      ...
```

⁴² Notice that the reported BidTime is a valid time whereas AskTime is not (rather than the opposite, as would be expected since only Ask is valid). This is apparently a bug in IQFeed's data: see <http://forums.iqfeed.net/index.cfm?page=topic&topicID=5594>

⁴³ IQFeed actually does report empty market-maker quotes, that have neither Bid nor Ask. Such IQFeed messages can be seen if you set the **Debug** parameter to true or 1 (see §12.1). However, *IQC* does not report such empty quotes by default, nor count them as valid "events" that should be checked against the requested **NumOfEvents** parameter value (default value: 10).

If your IQFeed account is not authorized/subscribed for Level 2 data, you will receive an error message whenever you request market depth data:⁴⁴

Account not authorized for Level II

If your IQFeed account is authorized for Level 2 data but not for a certain exchange, you will receive an error message when requesting market depth info from that exchange:

```
>> data = IQC('marketdepth', 'Symbol', 'IBM') % not subscribed to NYSE Level2
Error using IQC
Symbol 'IBM' was not found!
```

The following parameters affect market depth data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁴⁵	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: <ul style="list-style-type: none"> '@ES#' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0
NumOfEvents	integer	10 ⁴⁶	One of: <ul style="list-style-type: none"> inf – continuous endless streaming Level 2 data for specified symbol(s) N>1 – only process N incoming quotes 1 – get only a single quote 0 – stop streaming market depth data -1 – return the latest Level 2 data while continuing to stream new data updates
IncludeEmpty Quotes	logical (true/false)	false	If set to true or 1, empty Level 2 quotes (with neither a valid Bid nor valid Ask) will also be returned. By default (false), they will not be.
Timeout	number	5.0	Max number of seconds to wait (0-9000) for data in blocking mode (0 means infinite)



Note: Market Depth (Level 2) data is only available in the Professional *IQC* license.

⁴⁴ Old versions of *IQC* used to automatically try to establish a connection with IQFeed's L2 server during *IQC* startup, causing the error message to display for users without an IQFeed Level 2 subscription. This was confusing, since the L2 connection attempt during *IQC*'s startup was automatic, not caused by a user marketdepth request. In *IQC* version 2.17 (released on 2019-05-05) the mechanism was changed: *IQC* now connects to L2 server only as needed (first *IQC* marketdepth request).

⁴⁵ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

⁴⁶ The default value of **NumOfEvents** was changed from `inf` to 10 in *IQC* version 2.17 (released on 2019-05-05)

4.5 Greeks, fair value, and implied volatility

Extra data can be fetched (calculated) for asset options using the 'greeks' action:

- Greeks (*Delta, Vega, Theta, Rho, Gamma* etc.)
- Fair value for the derivative and the difference vs. actual trading price
- Implied volatility based on the fair vs. trading prices

```
>> data = IQC('greeks', 'symbol', 'IBM1814L116')
data =

    Symbol: 'IBM1814L116'
    Asset_Name: 'IBM DEC 2018 C 116.00'
    Strike_Price: 116
    Expiration_Date: '12/14/2018'
    Days_To_Expiration: 30
    Inferred_Asset_Side: 'Call'
    Underlying_Symbol: 'IBM'
    Underlying_Asset_Name: 'INTERNATIONAL BUSINESS MACHINE'
    Underlying_Spot: 121.3
    Underlying_Historic_Volatility: 37.1
    Assumed_Risk_Free_Rate: 0
    Assumed_Dividend_Yield: 0
    Asset_Fair_Value: 8.1193
    Asset_Latest_Price: 7.05
    Asset_Price_Diff: 1.0693
    Implied_Volatility: 0.28242
    Volatility_Used_By_Greeks: 0.371
    Delta: 0.68197
    Vega: 0.12404
    Theta: -0.076697
    Rho: 6.1318
    CRho: 6.7992
    Omega: 10.189
    Lambda: 10.189
    Gamma: 0.027646
    Vanna: -0.3527
    Charm: 0.0021809
    Vomma: 5.8043
    Veta: 2.4262
    Speed: -0.0012419
    Zomma: -0.061581
    Color: -0.00038078
    Ultima: -45.238
    Annual_Factor_Used: 365
    This_Asset_Latest_Quote: [1x1 struct]
    Underlying_Latest_Quote: [1x1 struct]
    This_Asset_Fundamentals: [1x1 struct]
    Underlying_Fundamentals: [1x1 struct]
```

The reported Matlab struct contains a few fields with basic information on the derivative and its underlying, followed by fair-value, implied volatility and Greek values.

At the bottom of the returned data-struct, four sub-structs provide direct access to the latest quotes data (§4.1, for example `data.This_Asset_Latest_Quote.Total_Volume`) and fundamenta data (§4.2, for example `data.Underlying_Fundamentals.Average_Volume`), for both the option asset and its underlying stock.

Note that the returned quotes data is subject to the **Fields** parameter value that was set in the most recent quotes data query (see §4.2). If the most recent **Fields** value does not include pricing data (`Most_Recent_Trade`, `Bid`, `Ask`, `Last`, or `Close` fields), then some returned data fields (for example `Asset_Price_Diff` and `Implied_Volatility`) will be empty.

The following Greek values are reported by *IQC*:

Field	Symbol	Derivative order	Definition	Description
Delta	Δ	1	$\partial V / \partial S$	Sensitivity of fair value to changes in the underlying asset's spot price
Vega	\mathbf{V}	1	$\partial V / \partial \sigma$	Sensitivity of fair value to changes in the underlying asset's volatility; also called Kappa
Theta	Θ	1	$-\partial V / \partial \tau$	Sensitivity of fair value to maturity time (decay)
Rho	ρ	1	$\partial V / \partial r$	Sensitivity of fair value to risk-free rate
CRho	-	1	$\partial V / \partial b$	Sensitivity of fair value to the carry-rate
Omega, Lambda	Ω λ	1	$\Delta \times S/V$	% change in fair value due to a 1% change in the underlying asset price (these are synonym fields, both are reported for convenience)
Gamma	Γ	2	$\partial \Delta / \partial S$	Sensitivity of Delta to changes in the underlying asset's spot price
Vanna	-	2	$\partial \Delta / \partial \sigma$	Sensitivity of Delta to changes in the underlying asset's volatility
Charm	-	2	$-\partial \Delta / \partial \tau$	Sensitivity of Delta to maturity time (decay)
Vomma	-	2	$\partial V / \partial \sigma$	Sensitivity of Vega to changes in underlying asset's volatility; also sometimes called Volga
Veta	-	2	$\partial V / \partial \tau$	Sensitivity of Vega to the maturity time
Speed	-	3	$\partial \Gamma / \partial S$	Sensitivity of Gamma to changes in the underlying asset's spot price
Zomma	-	3	$\partial \Gamma / \partial \sigma$	Sensitivity of Gamma to changes in the underlying asset's volatility
Color	-	3	$\partial \Gamma / \partial \tau$	Sensitivity of Gamma to maturity time (decay)
Ultima	-	3	$\partial^3 V / \partial \sigma^3$	Sensitivity of Vomma (Volga) to changes in the underlying asset's volatility

You can request data for multiple symbols at the same time, in a single *IQC* command, by specifying the symbols using a colon-delimited string or a cell-array. For example:

```
>> data = IQC('greeks', 'symbols', {'IBM1814L116', 'IBM1814X116'});
>> data = IQC('greeks', 'symbols', 'IBM1814L116:IBM1814X116'); % equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols:

```
data =
2x1 struct array with fields:
    Symbol
    Asset_Name
    Strike_Price
    ...
```

If you have Matlab's Parallel Computing Toolbox, set the **UseParallel** parameter to `true` (or 1) to process the Greeks query for the specified symbols in parallel (see §3.6):

```
>> data = IQC('greeks', 'symbols', {'IBM1814L116', 'IBM1814X116'}, ...
    'UseParallel', true);
```



Notes:

1. Greeks and related derivative data (the the 'greeks' action in general) are only available in *IQC* Professional and trial licenses, not in the Standard license.
2. Greeks, fair-price and implied vol values are computed by *IQC* on your local computer. They are **NOT** provided by IQFeed, and are **NOT** approved by DTN.
3. There is a performance impact: the calculations require some data fetches from IQFeed. These extra fetches and calculations may take up to 0.3-1 secs per query, depending on CPU, IQFeed round-trip latency, and the specific parameters.
4. The calculations assume vanilla European-style options using Black-Scholes-Merton's model.⁴⁷ Using *IQC*'s calculations with other derivatives (American/Asian/barrier/exotic options etc.) may result in incorrect/misleading values.
5. There are various possible ways to estimate implied volatility from the option's trading price and fair value. *IQC* uses a standard Newton-Raphson iterative approximation method; other methods may result in slightly different values.
6. Certain fields sometimes report invalid values. For example, `Implied_Volatility` may contain `-Inf` or `+Inf` when the Newton-Raphson algorithm fails to converge to a valid value. Likewise, some Greeks may contain a NaN value in certain cases (for example, a contract so far out-of-the-money that it has no trading price).
7. Some Greeks are also known by other names: *Vega* is sometimes called *Kappa*; *Vomma* is also known as *Volga* or *vega convexity*; *Omega* is also called *Lambda* or *elasticity*; *Charm* is also known as *delta decay*; and *Color* as *gamma decay*.
8. Various sources/systems calculate Greeks in different manners. For example, *Vega*, *Rho*, *Veta* and *Ultima* values are sometimes divided by 100 (but not in *IQC*); *Theta*, *Charm*, *Veta* and *Color* are sometimes annualized and sometimes divided by a representative number of days per year (365/364/360/253/252) to provide 1-day estimates (customizable in *IQC*, 365 by default);⁴⁸ The foreign rate/dividends yield is ignored by some sources and included by others in the calculations; Some sources report *Color* as the *positive* rate of change of *Gamma* relative to maturity time, while others report it as the *negative* rate of change.⁴⁹ In addition, some sources apparently have buggy math implementations.⁵⁰ The result is that different sources provide different Greek values for the same inputs. *IQC*'s values are basically identical to those of Matlab's Financial Toolbox, NAG and Maple.⁵¹ Unfortunately, IQFeed's standalone Option Chains utility reports different values. *IQC* adheres to the core math formulae⁵² and we believe that *IQC* provides mathematically-accurate results. However, the discrepancy between the values reported by different systems means that you must carefully ensure that *IQC*'s reported values fit your needs and expectations.

⁴⁷ Support for American options is planned in a future release of *IQC*; there are no current plans to support Asian/exotic options.

⁴⁸ Matlab's Financial Toolbox, NAG and Maple report annualized values; for annual values in *IQC*, set the **AnnualFactor** to 1.

⁴⁹ For example, the reported *Color* value is negative in NAG compared to *IQC* and Maple.

⁵⁰ This does not imply that there are no calculation bugs in *IQC*'s implementation; the Greeks calculation is not trivial.

⁵¹ Excluding a few quirks, such as a negative *Color* value reported by NAG, or Maple's *Lambda* calculation, or the **AnnualFactor** of 1 used by both NAG & Maple. Also compare the very similar values reported by the online calculator <http://option-price.com>

⁵² John Hull, *Options, Futures, and Other Derivatives* (ISBN 9780134472089); [https://en.wikipedia.org/wiki/Greeks_\(finance\)](https://en.wikipedia.org/wiki/Greeks_(finance))

By default, *IQC* uses the derivative's fundamental data and default 0% rates in its calculations. You can override these defaults using the following optional parameters:

- **UnderlyingSymbol** – by default this is the `Asset_Name`'s first string token, or the first portion of **Symbol**. For example, for IBM1814L116, `Asset_Name`='IBM DEC 2018 C 116.00' so `Underlying_Symbol` is set to 'IBM' (the first token in the `Asset_Name`); for @BOF20P28500 the `Underlying_Symbol` is set to '@BOF20' (Soybean Oil Jan 2020 Future).⁵³ To check the auto-inferred **UnderlyingSymbol**, check the `Underlying_Asset_Name` field in the returned data. The `Underlying_Symbol` value can be overridden using the **UnderlyingSymbol** parameter. For example, you could specify that the underlying symbol for Greeks computation of GOOG1816K1000 is not the default 'GOOG' (Alphabet Inc Class C), but rather 'GOOGL' (Class A).
- **Side** – by default, the option side ('Call' or 'Put') is determined by *IQC* from the derivative contract's `Asset_Name`. For example, for IBM1814L116, `Asset_Name`='IBM DEC 2018 C 116.00', which is automatically inferred to be a Call option. You can override the inferred side for contracts that have a non-standard `Asset_Name` (or one which is not properly reported by IQFeed in its Fundamental Data message) that *IQC* cannot properly analyze.
- **HistoricVolatility** – this is usually reported by IQFeed in the underlying asset's fundamental data (`data.Underlying_Fundamentals.Historical_Volatility`) and this is used in *IQC* by default. Instead of this reported value, you can specify another value (for example, the S&P 500 volatility), as a fixed percent value.
- **UseImpliedVolatility** – by default, *IQC* uses **HistoricVolatility** to calculate Greek values. Set **UseImpliedVolatility** to 1 or true to calculate Greeks using the `Implied_Volatility` instead (this may be useful for some commodities).
- **RiskFreeRate** – this is the domestic risk-free rate. *IQC* uses 0% by default; you can specify any other fixed percentage rate (based on e.g. LIBOR⁵⁴ or T-bill⁵⁵).
- **DividendsYield** – this is the underlying asset's dividend yield. *IQC* uses 0% by default; you can specify any other fixed percentage value. In the context of Forex currencies, this value may represent the foreign risk-free (carry) rate.
- **DaysToExpiration** – by default, *IQC* determines the duration until contract expiry (maturity) based on its `Expiration_Date`. This duration can be set to any positive number of days (not necessarily an integer value).
- **AnnualFactor** – by default, *IQC* normalizes the reported *Theta*, *Charm*, *Veta* and *Color* values by dividing the computed annualized value by 365 in order to provide 1-day estimates. You can override this scaling factor to any positive number. Setting a value of 1 provides annualized results (i.e., not 1-day estimates), as reported by Matlab's Financial Toolbox, NAG and Maple. For various uses you could also use other factors such as 364, 360, 253, 252, 12 or 4.

⁵³ Some short-listed future contracts do not have an immediately inferable **UnderlyingSymbol**. For example, the underlying symbol of @CF20C4000 (Corn Jan 2020 Call 4000) is not @CF20 (which does not exist). In such cases, *IQC* tries to use a corresponding contract of the next or the following months (in this case @CH20 - Corn March 2020). Sometimes this mechanism fails. For example, the underlying of @S2Z19C8700 (Soybeans Week 2 Dec 2019 Call 8700) is not @S2Z19, @S2F20 etc. (which do not exist) but rather @S2Z0 (Soybeans Jan 2020). In such cases, you must specify the **UnderlyingSymbol** manually.

⁵⁴ You can query the current LIBOR rate with *IQC*, for example using symbol ONLIB.X (overnight rate), 1MLIB.X (1 month), 3MLIB.X (3 months), or 1YLIB.X (1 year). Additional durations are also available (<http://iqfeed.net/symbolguide/index.cfm?pick=indexRATES&guide=mkindices>), but a 1-month rate is often used even for shorter or longer option durations, for consistency. Also see <http://forums.iqfeed.net/index.cfm?page=topic&topicID=4387>.

⁵⁵ You can query the current T-bill rate with *IQC*, for example using symbol TB30.X (30-day rate), IRX.XO (91 days), TB180.X (180 days), or 1YCMY.X (1 year). Also see <http://forums.iqfeed.net/index.cfm?page=topic&topicID=4387>.

Here is a usage example with some non-default parameters:

```
>> data = IQC('greeks', 'symbol', 'IBM1814L116', 'DaysToExpiration', 13.5, ...
              'RiskFreeRate', 2.5, 'DividendsYield', 3.2, 'AnnualFactor', 1)
```

The following parameters affect Greeks data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁵⁶	colon-delimited string, or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> 'GOOG1816K1000' 'IBM1814L116:GOOG1816K1000' {'IBM1814L116', 'GOOG1816K1000'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols will be done in parallel (see §3.6).
MaxWorkers	integer	(current parallel pool size)	Max number of parallel workers to use (up to the current pool size) when UseParallel =1
Underlying Symbol	string	" (i.e. taken from the contract name)	Symbol of the derivative's underlying asset
Side	string	" (i.e. taken from the contract name)	Either 'Call' or 'Put'
HistoricVolatility	number	-1 (i.e. taken from the underlying asset's reported historic volatility)	Value that represents the underlying's price volatility (in percent). 1.0 means 1%; -1 means a dynamic value based on the underlying asset's reported historic volatility.
UseImplied Volatility	logical (true/false)	false	If set to true or 1, the <i>Implied_Volatility</i> (not HistoricVolatility) will be used for Greeks
RiskFreeRate	number	0.0	Domestic risk-free rate Specified in percent; 1.0 means 1%.
DividendsYield	number	0.0	Underlying stock's dividends yield, or the foreign currency risk-free (carry) rate. Specified in percent; 1.0 means 1%.
DaysToExpiration	number	-1 (i.e. taken from the contract's expiration date)	Number of days until the contract expires (matures)
AnnualFactor	number	365	The computed <i>Theta</i> , <i>Charm</i> , <i>Veta</i> and <i>Color</i> values are divided by this factor before being reported. Typical values are 365, 364, 360, 253, 252, 12, 4 or 1.



Note: The Greeks functionality is only available in the Professional and trial *IQC* licenses, not in the Standard license.

⁵⁶ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

4.6 Market summary data and scanner

All the queries described so far in this chapter return data for individually-specified **Symbols**. We can retrieve a complete market snapshot of all traded securities of a specific **SecType** and **Exchange**, using a 'summary' query:

```
>> data = IQC('summary') % latest 5-minute data for all NYSE equities
data =
    4749x1 struct array with fields:
        Symbol
        Exchange
        Type
        Last
        TradeSize
        TradedMarket
        TradeDate
        TradeTime
        Open
        High
        Low
        Close
        ... (total of 28 data fields)
>> data(1)
ans =
    struct with fields:
        Symbol: 'A'
        Exchange: 7
        Type: 1
        Last: 72.5315
        TradeSize: 5
        TradedMarket: 5
        TradeDate: 20190711
        TradeTime: 103502
        Open: 73.77
        High: 73.78
        Low: 72.5315
        Close: 73.37
        Bid: 72.51
        BidMarket: 18
        BidSize: 200
        Ask: 72.54
        AskMarket: 5
        AskSize: 100
        Volume: 257497
        PDayVolume: 1785225
        UpVolume: 84799
        DownVolume: 80276
        NeutralVolume: 92422
        TradeCount: 2371
        UpTrades: 733
        DownTrades: 832
        NeutralTrades: 806
        VWAP: 73.1309
```

This query shows that 4749 equities are currently trading on NYSE⁵⁷. This data is daily (i.e., the cumulative day's Open/High/Low/Volume etc.), and is updated on IQFeed's servers every 5 minutes. DTN says that *“the timing of the snapshot is not guaranteed, but data will be gathered every 5 minutes”*. Therefore, you should assume for safety that the data is up to 5 minutes old. To get the latest data, use real-time snapshot (§4.1) and fundamental (§4.2) queries, or streaming quotes (§6.1).

⁵⁷ This query was run on July 11, 2019 at 11am EDT

The default **DataType** parameter value ('snapshot') fetches trading data. To fetch a market summary of fundamental data, set **DataType**='fundamental'. For example:

```
>> data = IQC('summary', 'DataType','fundamental')
data =
    4749x1 struct array with fields:
        Symbol
        Description
        PeRatio
        AvgVolume
        ...    (total of 41 data fields)

>> data(1)
ans =
    struct with fields:
        Symbol: 'A'
        Description: 'AGILENT TECHNOLOGIES'
        PeRatio: 21
        AvgVolume: 1908
        DivYield: 0.89
        DivAmount: 0.164
        DivRate: 0.656
        PayDate: 20190724
        ExDivDate: 20190701
        CurrentEps: 3.5
        SIC: 3825
        Precision: 4
        Display: 14
        GrowthPercent: -0.14
        FiscalYearEnd: 20181001
        Volatility: 16.5
        ListedMarket: 7
        OptionRoots: 'A'
        InstitutionalPercent: 86.905
        YearEndClose: 67.46
        Beta: 1.35
        Assets: 3848
        Liabilities: 1171
        BalanceSheetDate: 20190430
        LongTermDebt: 1799
        CommonSharesOutstanding: 315993
        MarketCap: 23184
        x52WeekHigh: 82.27
        x52WeekHighDate: 20190321
        x52WeekLow: 61.01
        x52WeekLowDate: 20181024
        CalHigh: 82.27
        CalHighDate: 20190321
        CalLow: 62
        CalLowDate: 20190103
        LastSplit: []
        LastSplitDate: []
        PrevSplit: []
        PrevSplitDate: []
        NAICS: 334516
        ShortInterest: 4130628
```

You can control the query using the **DataType** (default: 'snapshot'), **SecType** (default: 'equity') and/or **Exchange** (default: 'NYSE') parameters:

```
>> data = IQC('summary', 'SecType','bond', 'Exchange','NYSE', ...
               'DataType','fundamental');
```

```
>> struct2table(data)
ans =
    6326x11 table
    Symbol      Description      Precision Display ListedMarket MaturityDate CouponRate ...
    'A20.CB'     'AGILENT TECHNOLOGIES INC. 5.0% SR NTS' [] 12 7 20200715 [] ...
    'A23.CB'     'AGILENT TECHNOLOGIES INC. 3.875% 07/15/2' [] 12 7 20230715 3.875
    'A26.CB'     'AGILENT TECHNOLOGIES INC 3.05 09/22/2026' [] 12 7 20260922 3.050
    'AA28.CB'    'ALUMINUM CO OF AMERICA 6.75% NTS 1/15/28' [] 12 7 20280115 []
    'AA20.CB'    'ALCOA INC. 6.15% SR NTS' [] 12 7 20200815 []
    'AA21.CB'    'ALCOA INC. 5.4% SR NTS' [] 12 7 20210415 []
    'AA22.CB'    'ALCOA INC NT 5.87%' [] 12 7 20220223 []
    'AA24.CB'    'ALCOA INC 5.125% 10/01/2024' [] 12 7 20241001 []
    'AA27.CB'    'ALCOA INC 5.9% NTS 2/1/27' [] 12 7 20270201 5.900
    'AA37.CB'    'ALCOA INC 5.95% NTS 2/1/37' [] 12 7 20370201 5.950
    'AAP20.CB'   'ADVANCE AUTO PARTS INC. 5.75%' [] 12 7 20200501 []
    'AAP22.CB'   'ADVANCE AUTO PARTS INC 4.5%' [] 12 7 20220115 []
    'AAP23.CB'   'ADVANCE AUTO PARTS 4.5 12/01/23' 2 12 7 20231201 []
    'AAPL22.CB'  'APPLE INC 1.00% NOTES 22' [] 0 7 20221110 []
    ...
```

Note that there is no **Symbol** parameter in a 'summary' query – data for all the symbols that match the specified **SecType** and **Exchange** is returned. For historic market summaries, add the **Date** parameter (see §5.6).

By default, only data fields that contain information are returned. For example, in the snapshot query for equities, only 28 of 35 data fields are reported; 7 fields are removed from the returned struct array since they contain an empty ([]) value for all securities:

MutualDiv, SevenDayYield, OpenInterest, Settlement, SettlementDate, ExpirationDate, Strike.⁵⁸ Similarly, 3 additional fields (High, Low and VWAP) are not reported for bonds (only 25 fields contain information). To include all these fields (with their empty data values) in the reported data, set the **ReportEmptyFields** parameter to true (or 1):

```
>> data = IQC('summary', 'ReportEmptyFields',true)
data =
    4749x1 struct array with fields:
        Symbol
        Exchange
        ...      (total of 35 data fields)

>> data(1)
ans =
    struct with fields:
        Symbol: 'A'
        ...
        NeutralTrades: 806
        VWAP: 73.1309
        MutualDiv: []
        SevenDayYield: []
        OpenInterest: []
        Settlement: []
        SettlementDate: []
        ExpirationDate: []
        Strike: []
```

Likewise, with a 'fundamental' query, only 41 of 47 possible fields are reported for equities (EstEps, MaturityDate, CouponRate, LEAPs, WRAPs and Expiration fields are not reported);⁵⁹ for bonds only 11 fields are reported (Symbol, Description, Precision, Display, ListedMarket, MaturityDate, CouponRate, x52WeekHigh, x52WeekHighDate, x52WeekLow and x52WeekLowDate), while 36 other fields are not. As before, to include these fields (with their empty data values) in the reported data, set **ReportEmptyFields** to true (or 1).

⁵⁸ Additional fields (for example, Open, High, Low) are missing when the query is run early in the day, before start of trading.

⁵⁹ The EstEps field was reported by IQFeed in some runs but not others; the reason for this is unclear.



Market summary queries can take a long time to download data, depending on amount of data and your computer speed. To ensure the query does not time-out before completing the download, the default **Timeout** value for summary queries is set to 300 secs, unlike other queries (5 secs). In some cases, you may need to specify an even larger **Timeout**.

To reduce processing time, numeric codes (e.g., `Exchange`, `TradedMarket` and `NAICS`) are not interpreted into textual form, unlike the corresponding real-time snapshot (§4.1) and fundamental (§4.2) queries. Use a lookup query (§8) to fetch a description of such codes.

The returned data can be filtered based on multiple criteria, effectively serving as a market scanner (for the latest data by default, or for any other historic date). This is done by setting the **Filter** parameter to the relevant criteria. For example, to return all NYSE equities whose latest market capitalization is larger than \$5Bn, we set a condition on the `MarketCap` data field (which reports values in \$Mn, so \$5Bn=5000):

```
data = IQC('summary', 'SecType', 'Equity', 'DataType', 'fundamental', ...
          'Exchange', 'NYSE', 'Filter', 'MarketCap>5000');
```

This query only returns 1398 equities, compared to the unfiltered 4749. Multiple filter criteria can be specified using a cell-array. For example (this returns just 100 equities):

```
data = IQC('summary', 'SecType', 'Equity', 'DataType', 'fundamental', ...
          'Exchange', 'NYSE', 'Filter', {'MarketCap>5000', 'PeRatio<9'});
```

Multiple filter criteria are AND'ed, meaning that all of the criteria conditions must be met for a security to be reported. If you want to use an OR condition (for example, Market-cap>\$5B **or** P/E<9), combine the conditions within a single filter criterion:

```
data = IQC('summary', 'SecType', 'Equity', 'DataType', 'fundamental', ...
          'Exchange', 'NYSE', 'Filter', 'MarketCap>5000 | PeRatio<9');
```

In general, any Matlab expression (arithmetic, function etc.) that involves the reported data field(s) and results in a scalar logical value, is acceptable as **Filter** criteria.

Criteria conditions are case-sensitive and must use exactly the same field names as the reported data fields, otherwise the criteria will be ignored. For example:

```
data = IQC('summary', 'SecType', 'Equity', 'DataType', 'fundamental', ...
          'Exchange', 'NYSE', 'Filter', 'marketcap>5000');

Warning: ignoring bad summary filter 'marketcap>5000': Unrecognized function
or variable 'marketcap'
(Type "warning off IQC:summary:filter" to suppress this warning.)
```

Whenever any field is included in a filter condition, securities that do not have data for that field (have an empty [] value) will automatically be filtered out of the returned data – they are considered to have failed the entire condition, even if it was only a part of an **or** condition. For example, if your filter condition is `MarketCap>5000`, then securities that have `MarketCap=[]` (i.e. unknown) will not be reported.

Note that the filtering/scanning is not applied at the data source (IQFeed) but rather in *IQC*, after the full set of data has been downloaded from IQFeed. Therefore, filtered queries will always take longer to process than regular (unfiltered) summary queries.

Note: Market summaries are only available with IQFeed client 6.1 or newer, and only if you are subscribed to the requested data at DTN and there is a relevant **data** download (summary data is only available for some combinations of **SecType** and **Exchange**). In all other cases, you may receive an error such as one of the following:

The 'summary' query is only supported by IQFeed client 6.1 or newer; you are using version 6.0.

IQC market summary query error: 50004,User not authorized for market summary file requested.

IQC market summary query error: 50007,No file available.

A related mechanism for fetching pre-filtered market scans for a select number of **Exchanges** and **Filters** is available by setting **DataType**='Top'. This scanner type does *not* depend on DTN subscription or IQFeed 6.1, and is *much* faster than snapshot queries. However, it is limited to just 13 predefined filters and 3 US exchanges (AMEX, NYSE, NASDAQ), only supports equities, does not provide historic data, and only returns only up to top 50 matching equity symbols⁶⁰ with a few relevant data fields (far fewer fields than the snapshot queries), updated every 5 minutes during the trading day:

```
>> data = IQC('summary', 'DataType','top', 'Exchange','NYSE', 'Filter','active')
data =
  50x1 struct array with fields:
      ...

>> data(1)
ans =
  struct with fields:
      Last_Update_Time: '2019/07/12 16:25 EST'
      Exchange: 'NYSE'
      Symbol: 'ABEV'
      Company_Name: 'AMBEV S.A.'
      Last_Price: 4.84
      Previous_Price: 4.95
      Price_Change_Dollars: -0.11
      Price_Change_Percent: -2.222
      Last_Volume: 50633779
      Previous_Volume: 27402071
      Volume_Change_Percent: 84.78
```

Several **Filter** types return additional data fields, depending on the filter. For example:

```
>> data = IQC('summary', 'DataType','top', 'Filter','volume spike');
>> data(1)
ans =
  struct with fields:
      Last_Update_Time: '2019/07/16 16:25 EST'
      Exchange: 'NYSE'
      Symbol: 'CPE'
      Company_Name: 'CALLON PETROLEUM'
      Last_Price: 5.73
      Previous_Price: 5.38
      Price_Change_Dollars: 0.35
      Price_Change_Percent: 6.506
      Last_Volume: 55574971
      Previous_Volume: 57244474
      Volume_Change_Percent: -2.92
      Average_Volume: 8296000
      Volume_vs_Avg_Change_Percent: 569.9
```

⁶⁰ The reported equities must not only match the predefined **Filter** condition, but also have a close price > \$2 and volume > 0.

Here's another example – top NASDAQ equities with last price lower than their VWAP:

```
>> data = IQC('summary', 'DataType','top', 'Exchange','NASDAQ', ...
              'Filter','vwap % down');

>> data(1)
ans =
    struct with fields:
        Last_Update_Time: '2019/07/16 16:25 EST'
        Exchange: 'NASDAQ'
        Symbol: 'IMRN'
        Company_Name: 'IMMURON LTD ADR'
        Last_Price: 4.3
        Previous_Price: 2.93
        Price_Change_Dollars: 1.37
        Price_Change_Percent: 46.758
        Last_Volume: 11022919
        Previous_Volume: 4704
        Volume_Change_Percent: 234230.76
        VWAP: 5.427
        Last_Minus_VWAP_Chng_Pct: -20.77
```

Note that the 'Top' query may return empty ([]) data for some combinations of **Exchange** and **Filter** on certain dates, depending on the market data and availability of the requested scanner on IQFeed's servers. ⁶¹ Also note that the reported data may be up to 5 minutes old during the trading day (depending on the query time).

The following table summarizes the differences between market summary query types:

	Snapshot	Fundamental	Top
DataType	'snapshot'	'fundamental'	'top'
Exchange	Multiple		Only AMEX, NYSE, NASDAQ
SecType	Multiple		Only 'equity'
Date	Latest (intra-day) or historic (end of day)		Only latest (updated every 5 minutes during the trading day)
Filter type	Any Matlab function of any combination of data fields		Only one of 13 predefined types
Filter combinations	Multiple criteria supported		Not supported
Processing time	Tens/hundreds of secs		<1 sec
IQFeed client	Requires client 6.1 or newer		Any (client 5.0 or newer)
IQFeed connection	Requires an active IQFeed connection (<i>IQConnect</i> login)		Does not require an active IQFeed connection
Result data fields	Up to 35 fields	Up to 47 fields	Only 11-13 data fields
Result records	All securities that fit the parameters (many thousands)		Only the top 50 securities

⁶¹ As of December 2019, IQFeed has a server problem causing AMEX data to return no results. DTN is working on a fix for this.

The following parameters affect market summary data queries:

Parameter	Data type	Default	Description
DataType	string	'snapshot'	Either 'snapshot', 'fundamental' or 'top'
Exchange	string	'NYSE'	One of the markets listed in §8.3
SecType	string	'Equity'	One of the security types listed in §8.4. SecType is ignored when DataType ='top'
Date	integer or string or datetime object	now (latest available data)	Date for which to fetch the end-of-day data. Examples: <ul style="list-style-type: none"> • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' Date is ignored when DataType ='top'
ReportEmptyFields	logical	false or 0	If true, then irrelevant data fields (which contain empty [] values for all securities) are reported; if false (default), they are not
Filter	string	'active'	When DataType ='top', one of: <ul style="list-style-type: none"> • 'active' – most active (highest volume) • 'gainer' – highest positive price \$ change • 'loser' – lowest negative price \$ change • '% gainer' – highest pos. price % change • '% loser' – lowest neg. price % change • '52 week high' – daily high > 52-week • '52 week low' – daily low < 52-week • 'volume up' – compared to previous • 'volume spike' – compared to average • 'VWAP up' – last price > VWAP • 'VWAP down' – last price < VWAP • 'VWAP % up' – % above VWAP • 'VWAP % down' – % below VWAP
	string or cell-array of strings	{ }	When DataType ='snapshot', 'fundamental': Zero or more filter criteria (condition strings) – Matlab expression(s) involving the reported data fields, which result in a logical (true/false) value. Examples: <ul style="list-style-type: none"> • 'MaturityDate > 20241231' • 'MarketCap > 5000 & PeRatio < 9' • {'MarketCap > 5000', 'Beta >= 1.2'}
Timeout	number	300	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)



Note: market summary functionality is only available in the Professional IQC license

5 Historical and intra-day data

Historical data can be retrieved via the 'history' action, subject to your account subscription rights, and IQFeed's pacing limitations. Several data-types are available, which can be set using the **DataType** parameter (default: 'day').⁶²

5.1 Daily data

To retrieve historic daily data bars, set **DataType** to 'd' or 'day' (or just leave this parameter out, since 'day' is the default data type), and set the asset's **Symbol**:

```
>> data = IQC('history', 'symbol', 'IBM');
>> data = IQC('history', 'symbol', 'IBM', 'dataType', 'day') %equivalent
data =
100x1 struct array with fields:
    Symbol
    Datestamp
    Datenum
    High
    Low
    Open
    Close
    PeriodVolume
    OpenInterest
```

We received an array of Matlab structs containing daily bars, one per each of the last N trading days (**excluding** currently-trading day's bar for IQFeed clients 6.0 or older; **including** the current day's bar for 6.1 or newer). By default, we receive up to N=100 data bars, ordered from oldest to newest. We ran the query above using IQFeed client 5.2 on March 6, 2018 so we received daily data from 2017-10-10 until 2018-03-05:

```
>> data(1)
ans =
    Symbol: 'IBM'
    Datestamp: '2017-10-10'
    Datenum: 736978
    High: 148.95
    Low: 147.65
    Open: 147.71
    Close: 148.5
    PeriodVolume: 4032601
    OpenInterest: 0

>> data(end)
ans =
    Symbol: 'IBM'
    Datestamp: '2018-03-05'
    Datenum: 737124
    High: 157.49
    Low: 153.75
    Open: 154.12
    Close: 156.95
    PeriodVolume: 3670630
    OpenInterest: 0
```

You can aggregate the numeric values into Matlab arrays as follows:

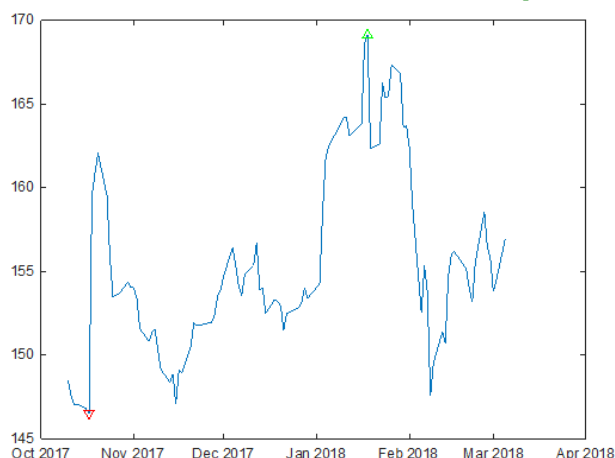
```
dates = {data.Datestamp}; % cell-array of strings
closes = [data.Close]; % array of numeric values
```

You can then use these arrays for vectorized processing, plotting etc. For example:

```
dates2 = datetime(dates); % array of datetime objects
[maxVal, maxIdx] = max(closes); % maximal value and location index
[minVal, minIdx] = min(closes); % minimal value and location index
```

⁶² <http://iqfeed.net/dev/api/docs/HistoricalviaTCPIP.cfm>


```
plot(dates2, closes); hold on;
plot(dates2(maxIdx), maxVal, '^g'); % maximal data point - green ▲
plot(dates2(minIdx), minVal, 'vr'); % minimal data point - red ▼
```



You can change the order at which the data bars are reported, using the **DataDirection** parameter (1 means oldest-to-newest (default); -1 means newest-to-oldest):

```
>> data = IQC('history', 'symbol', 'IBM', 'dataDirection', -1);
>> data(1)
ans =
    Symbol: 'IBM'
    Datestamp: '2018-03-05'
    Datenum: 737124
    High: 157.49
    Low: 153.75
    Open: 154.12
    Close: 156.95
    PeriodVolume: 3670630
    OpenInterest: 0
```

It is possible that there may be fewer than N=100 daily bars for an asset. For example, the symbol @EMF19 (1-month Euro-Dollar Jan 2019 future on CME) started trading on 2018-01-12, so we only get 35 daily bars when we run the query on 2018-03-06:

```
>> data = IQC('history', 'symbol', '@EMF19');
data =
    35x1 struct array with fields:
        Symbol
        ...
```

You can ask IQFeed to limit the maximal number of data bars (N) using the **MaxItems** parameter:

```
>> data = IQC('history', 'symbol', 'IBM', 'maxItems', 20)
data =
    20x1 struct array with fields:
        Symbol
        ...
```

In this example, `data(1).Datestamp='2018-02-05'`, i.e. 20 trading days ago.

Note that the **MaxItems** parameter only has an effect if the additional data bars actually exist. In other words, it controls the *maximum* number of returned data bars – the *actual* number of bars may be less than this value.⁶³

⁶³ For example, IQFeed's trial account is limited to 1-year of daily data points; IQFeed automatically trims trial-account queries down to this limit: <http://forums.dtn.com/index.cfm?page=topic&topicID=5535>

When the number of data bars that IQFeed sends is very large, it could take a while for the information to be sent. In such a case, *IQC* might time-out on the request and return only partial data. Such a case is detected and reported by *IQC*:

```
>> data = IQC('history', 'symbol','IBM', 'maxItems',-1)
Warning: IQC timeout: only partial data is returned: the Timeout parameter
should be set to a value larger than 5
data =
    1274x1 struct array with fields:
        Symbol
        ...
```

As suggested by the message, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather the data before returning the results:

```
>> data=IQC('history', 'symbol','IBM', 'maxItems',-1, 'timeout',60) %oldest:1/2/96
data =
    5577x1 struct array with fields:
        Symbol
        ...
```

You can also specify a **BeginDate/EndDate** interval for the returned data. Dates can be specified in several formats: numeric Matlab datenum (737089), Matlab datetime object, numeric yyymmdd (20180129), string ('2018/01/29', '2018-01-29', '20180129'). Note that **MaxItems** takes precedence over **BeginDate**, regardless of **DataDirection**. For example, if **MaxItems**=5, you will only get the 5 latest bars, for any **BeginDate**.⁶⁴

You can request historical data for multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('history', 'symbol',{'IBM','GOOG','AAPL'}, 'maxItems',20)
>> data = IQC('history', 'symbol','IBM:GOOG:AAPL', 'maxItems',20) %equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols (3 symbols with 20 data-points each, in this example):

```
data =
    20x3 struct array with fields:
        Symbol
        ...
>> data(1,2) % 2nd index (column) is the symbol; GOOG data is in data(:,2)
ans =
    struct with fields:
        Symbol: 'GOOG'
        Datestamp: '2018-07-10'
        Datenum: 737251
        High: 1159.59
        Low: 1149.59
        Open: 1156.98
        Close: 1152.84
        PeriodVolume: 798412
        OpenInterest: 0
```

In certain cases, when you request historic data for multiple symbols, you might receive a different number of data bars for different symbols, depending on data availability. In such cases, the result will not be an N-by-M struct array, but a cell array (one cell for each symbol) that contains struct arrays. For example:

```
data =
    1x3 cell array
        {77x1 struct}    {100x1 struct}    {55x1 struct}
```

⁶⁴ Note: Regular IQFeed accounts have access to 15+ years of daily data, but IQFeed limits its trial account to just 365 days of historical daily data – see <https://help.dtniq.com/support-faqs>

IQC queries for multiple symbols or dates (if **BeginDate** and **EndDate** are specified) can be parallelized using the **UseParallel** parameter, if you have a Professional *IQC* license and Matlab's Parallel Computing Toolbox (§3.6):

```
>> data = IQC('history', 'UseParallel',true, 'symbol',symbols) %multi-symbols
>> data = IQC('history', 'UseParallel',true, 'symbol','IBM',...
    'BeginDate',19900102, 'EndDate',20181028) %date range
```

By default, *IQC* reports 9 data fields for each daily history bar: Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, and OpenInterest. If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary):

```
>> data = IQC('history', 'fields',{})
data =
    struct with fields:
        CurrentFields: {1x9 cell}
        AvailableFields: {1x9 cell}

>> data.AvailableFields
ans =
    1x9 cell array
    Columns 1 through 6
        {'Symbol'} {'Datestamp'} {'Datenum'} {'High'} {'Low'} {'Open'}
    Columns 7 through 9
        {'Close'} {'PeriodVolume'} {'OpenInterest'}
```

If you have the Professional (or trial) *IQC* license, you can request *IQC* to report fewer data fields, and/or change the reported fields order, using the optional **Fields** parameter. **Fields** can be set to any subset of `AvailableFields`, as either a cell-array of strings, or as a comma-separated string. All subsequent daily history queries will report the requested fields, in the specified order, until **Fields** is changed again. For example:

```
>> data = IQC('history', 'Symbol','IBM', 'Fields',{'Datenum','Close'})
>> data = IQC('history', 'Symbol','IBM', 'Fields','Datenum,Close') %equivalent
data =
    100x1 struct array with fields:
        Datenum
        Close

>> data(1)
ans =
    struct with fields:
        Datenum: 737751
        Close: 134.34
```

The order of the specified **Fields** indicates the order in which the data fields will be reported. For example, to change the order of the reported data fields above:

```
>> data = IQC('history', 'Symbol', 'IBM', 'Fields','Close,Datenum ')
data =
    100x1 struct array with fields:
        Close
        Datenum
```

As noted, **Fields** can be set to any subset of the `availableFields`. If a bad field is specified (one which is not available in IQFeed), an error message will be displayed:

```
>> data = IQC('history', 'Symbol','IBM', 'Fields','Close, xyz')
Error using IQC
Bad field 'xyz' was requested in IQC history command (check the spelling).
Available fields are: Symbol,Timestamp,Datenum,High,Low,Open,Close,...
```

As noted above, whenever we change the set of fields (or even just their order), this new set of fields is used in all subsequent daily history queries in the current Matlab session.

To revert the reported set of fields to the default set (`AvailableFields`), set **Fields** to 'All' (or 'all'):

```
>> data = IQC('history', 'Symbol','IBM', 'Fields','all')
data =
    100x1 struct array with fields:
        Symbol
        Datestamp
        Datenum
        High
        Low
        Open
        Close
        PeriodVolume
        OpenInterest
```



Note: there are several important differences between the **Fields** parameter in history queries and in quotes queries (§4.1):

1. The **Symbol** field is not mandatory in history queries, and does not necessarily need to be the first reported data field, unlike in quotes queries.
2. *IQC* reports all available history data fields by default. You can use the **Fields** parameter to reduce the reported data fields, even down to just a single data field. In contrast, quotes queries report only some of the available fields by default.

The fewer fields that you request, the faster the processing time and the smaller the memory usage. To improve run-time performance and reduce memory consumption, request only those data fields that are actually needed by your program.

The following parameters affect daily history data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁶⁵	colon or comma-delimited string or cell-array of strings	(none)	Limits query to specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).
DataDirection	integer	1 meaning oldest first, newest last	Sets the order of data bars in the returned struct array. One of the following values: <ul style="list-style-type: none"> • 1 means oldest-to-newest (default) • -1 means newest-to-oldest
MaxItems	integer	100	Returns up to the specified number of data bars (if available). -1 means all available.

⁶⁵ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
BeginDate	integer or string or datetime object	'1900/01/01' (i.e., from as early as data is available)	<p>Earliest bar date. Examples:</p> <ul style="list-style-type: none"> • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' <p>Note: MaxItems has precedence over BeginDate: If there are more data points than MaxItems between BeginDate–EndDate, only the last MaxItems data points (from EndDate backward) will be returned, regardless of BeginDate.</p>
EndDate	integer or string or datetime	'2099/12/31' (i.e., until now)	<p>Latest bar date.</p> <p>See BeginDate parameter above for details.</p>
Timeout	number	5.0	Max # of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols or dates will be done in parallel (see §3.6; Professional <i>IQC</i> license only).
MaxWorkers	integer	(the current parallel pool size, up to 15)	Maximal number of parallel workers to use (up to the current pool size) when UseParallel =true
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, OpenInterest'	<p>Sets the list of data fields reported by <i>IQC</i> for each data bar, as a sub-set of IQFeed's default set of 9 fields.</p> <p>If Fields is set to an empty value ({} or ""), the list of current, available fields is returned.</p> <p>If Fields is not empty, subsequent history queries in the same Matlab session will return the specified fields, in the specified order (Professional <i>IQC</i> license only).</p> <p>Examples:</p> <ul style="list-style-type: none"> • 'Datestamp:Open:Close' • 'Datestamp,Open,Close' • {'Datestamp', 'Open', 'Close'} • 'All' (indicates all available fields)
Progress	string	" (empty string)	When Progress is set to 'console', the data download progress is displayed in the console. See §5.4 for details.

5.2 Weekly data

To retrieve historic weekly data bars, set **DataType** to 'w' or 'week':

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'week')
data =
    100x1 struct array with fields:
        Symbol
        Datestamp
        Datenum
        High
        Low
        Open
        Close
        PeriodVolume
        OpenInterest
```

As with the daily bars, we received an array of Matlab structs containing weekly bars, one per each of the last N weeks (**excluding** currently-trading day for IQFeed clients 6.0 or older; **including** the current day for 6.1 or newer). By default we receive up to N=100 data bars (~2 years), ordered from oldest to newest. We ran the query above on Tuesday March 6, 2018 using IQFeed client 5.2 so we received weekly data from Friday 2016-04-15 (the data bar for April 11-15, 2016) until 2018-03-05 (the data bar for Monday March 5, 2018 only, excluding March 6). Each bar's `Datestamp` indicates the end-date of the bar. Note that all data bars except for the latest have a Friday date.

As with the daily bars, you can change the data bars order, using the **DataDirection** parameter (1 means oldest-to-newest (default); -1 means newest-to-oldest).

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'week', 'dataDirection', -1);
```

As with the daily bars, you can ask IQFeed to limit the maximal number of data bars (N) using the **MaxItems** parameter:

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'week', 'maxItems', 20);
```

In this example, `data(1).Datestamp='2017-10-27'`, i.e. the Friday 20 weeks ago.

As with the daily bars, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather data before returning the results. This is typically not necessary for weekly data requests, because of the relatively small amount of data.

You can also specify a **BeginDate** for the returned data. Dates can be specified in various formats: as a numeric Matlab datenum (737089), a Matlab `datetime` object, numeric `yyyymmdd` (20180129), or as a string ('2018/01/29', '2018-01-29', '20180129').⁶⁶

For example, if we a query with a **BeginDate** of Monday Jan 29, 2018, we will receive data bars starting on Friday Feb 2, 2018 (which includes the weekly data of Jan 29):

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'week', 'BeginDate', 20180129);
```



Note: **unlike** daily data requests, you cannot specify **EndDate** in a request for historic weekly data bars. All weekly data bars up to yesterday/today⁶⁷ will be returned.

Also note that **MaxItems** has precedence over **BeginDate**, regardless of **DataDirection**. For example, if **MaxItems**=5, we'll only get the 5 latest bars, regardless of **BeginDate**.

⁶⁶ Note: Regular IQFeed accounts can access 15+ years of historic data, but IQFeed limits trial accounts to just one year – see <https://help.dtniq.com/support-faq>. Also note that in some cases, depending on current day-of-week compared to the requested **BeginDate**, an additional (older) bar might be returned that includes the week that was prior to the requested **BeginDate**.

⁶⁷ Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

As with daily data requests, you can request historical data for multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('history', 'symbol',{'IBM','GOOG','AAPL'}, ...
              'dataType','week', 'maxItems',20)

>> data = IQC('history', 'symbol','IBM:GOOG:AAPL', ...
              'dataType','week', 'maxItems',20) %equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols (3 symbols with 20 data-points each, in this example):

```
data =
    20x3 struct array with fields:
        Symbol
        Datestamp
        ...
```

In certain cases, when you request historic data for multiple symbols, you might receive a different number of data bars for different symbols, depending on data availability. In such cases, the result will not be an N-by-M struct array, but a cell array (one cell for each symbol) that contains struct arrays. For example:

```
data =
    1x3 cell array
    {77x1 struct}    {100x1 struct}    {55x1 struct}
```

IQC queries for multiple symbols can be parallelized using the **UseParallel** parameter, if you have a Professional *IQC* license and Matlab's Parallel Computing Toolbox (§3.6):

```
>> data = IQC('history', 'symbol',symbols, 'UseParallel',true, ...
              'dataType','week', 'maxItems',20)
```

By default, *IQC* reports 9 data fields for each weekly history bar: Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, and OpenInterest. If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary). If you have the Professional (or trial) *IQC* license, you can request *IQC* to report fewer data fields, and/or change the reported fields order, using the optional **Fields** parameter. All subsequent weekly history queries will report only the requested fields, in the specified order. Fewer fields mean faster processing time and smaller memory usage. Refer to §5.1 for a description of the **Fields** parameter usage.

The following parameters affect weekly history data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁶⁸	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM','AAPL','GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).

⁶⁸ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
DataDirection	integer	1 meaning oldest bar is first, newest is last	Sets the order of data bars in the returned struct array. One of the following values: <ul style="list-style-type: none"> • 1 means oldest-to-newest (default) • -1 means newest-to-oldest
MaxItems	integer	100	Returns up to the specified number of data bars (if available). -1 means all available.
BeginDate	integer or string or datetime object	'1900/01/01' (i.e., from as early as data is available)	<p>Earliest bar that includes a date. Examples:</p> <ul style="list-style-type: none"> • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' <p>Note: MaxItems has precedence over BeginDate: If there are more data points than MaxItems between BeginDate and yesterday/today⁶⁹, only the last MaxItems data points (from yesterday/today backward) will be returned, regardless of BeginDate.</p>
Timeout	number	5.0	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols will be done in parallel (see §3.6; Professional IQC license only).
MaxWorkers	integer	(the current parallel pool size, up to 15)	Max number of parallel workers to use (up to the current pool size) when UseParallel =1
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, OpenInterest'	<p>Sets the list of data fields reported by IQC for each data bar, as a sub-set of IQFeed's default set of 9 fields.</p> <p>If Fields is set to an empty value ({} or ''), the list of current, available fields is returned.</p> <p>If Fields is not empty, subsequent history queries in the same Matlab session will return the specified fields, in the specified order (Professional IQC license only).</p> <p>Examples:</p> <ul style="list-style-type: none"> • 'Datestamp:Open:Close' • 'Datestamp,Open,Close' • {'Datestamp', 'Open', 'Close'} • 'All' (indicates all available fields)

⁶⁹ Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

5.3 Monthly data

To retrieve historic monthly data bars, set **DataType** to 'm' or 'month':

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'month')
data =
    100x1 struct array with fields:
        Symbol
        Datestamp
        Datenum
        High
        Low
        Open
        Close
        PeriodVolume
        OpenInterest
```

As with the daily bars, we received an array of Matlab structs containing monthly bars, one per each of the last N months (**excluding** currently-trading day for IQFeed clients 6.0 or older; **including** the current day for IQFeed clients 6.1 or newer). By default we receive up to N=100 data bars (~8 years), ordered from oldest to newest. We ran the example query above on March 6, 2018 using IQFeed client 5.2 so we received monthly data from 2009-12-31 (the data bar for 12/2009) until 2018-03-05 (the data bar for March 2018 up to March 5, 2018, excluding data from March 6).

As with the daily bars, you can change the data bars order, using the **DataDirection** parameter (1 means oldest-to-newest (default); -1 means newest-to-oldest).

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'month', ...
              'dataDirection', -1);
```

As with the daily bars, you can ask IQFeed to limit the maximal number of data bars (N) using the **MaxItems** parameter:

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'month', 'maxItems', 20);
```

In this example, `data(1).Datestamp='2016-08-31'`, i.e. 20 months ago.

As with the daily bars, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather data before returning the results. This is typically not necessary for monthly data requests, because of the relatively small amount of data.

You can also specify a **BeginDate** for the returned data. Dates can be specified in various formats: as a numeric Matlab datenum (737089), a Matlab `datetime` object, numeric `yyyymmdd` (20180129), or as a string ('2018/01/29', '2018-01-29', '20180129').

For example, if we a query with a **BeginDate** of Jan 29, 2018, we will receive data bars starting on Jan 31, 2018 (which includes the monthly data of Jan 29):

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'month', 'BeginDate', 20180129);
```



Note: **unlike** daily data requests, you cannot specify **EndDate** in a request for historic monthly data bars. All monthly data bars up to yesterday/today⁷⁰ will be returned.

Also note that **MaxItems** has precedence over **BeginDate**, regardless of **DataDirection**. For example, if **MaxItems**=5, we'll only get the 5 latest bars, regardless of **BeginDate**.⁷¹

⁷⁰ Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

⁷¹ Note: Regular IQFeed accounts can access 15+ years of historic data, but IQFeed limits trial accounts to just one year – see <https://help.dtniq.com/support-faqs>

As with daily data requests, you can request historical data for multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('history', 'symbol',{'IBM','GOOG','AAPL'}, ...
    'dataType','month', 'maxItems',20)

>> data = IQC('history', 'symbol','IBM:GOOG:AAPL', ...
    'dataType','month', 'maxItems',20) %equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols (3 symbols with 20 data-points each, in this example):

```
data =
    20x3 struct array with fields:
        Symbol
        Datestamp
        ...
```

In certain cases, when you request historic data for multiple symbols, you might receive a different number of data bars for different symbols, depending on data availability. In such cases, the result will not be an N-by-M struct array, but a cell array (one cell for each symbol) that contains struct arrays. For example:

```
data =
    1x3 cell array
    {77x1 struct}    {100x1 struct}    {55x1 struct}
```

IQC queries for multiple symbols can be parallelized using the **UseParallel** parameter, if you have a Professional *IQC* license and Matlab's Parallel Computing Toolbox (§3.6):

```
>> data = IQC('history', 'symbol',symbols, 'UseParallel',true, ...
    'dataType','month', 'maxItems',20)
```

By default, *IQC* reports 9 data fields for each monthly history bar: Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, and OpenInterest. If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary). If you have the Professional (or trial) *IQC* license, you can request *IQC* to report fewer data fields, and/or change the reported fields order, using the optional **Fields** parameter. All subsequent monthly history queries will report only the requested fields, in the specified order. Fewer fields mean faster processing time and smaller memory usage. Refer to §5.1 for a description of the **Fields** parameter usage.

The following parameters affect monthly history data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁷²	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM','AAPL','GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).

⁷² In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
DataDirection	integer	1 meaning oldest bar is first, newest is last	Sets the order of data bars in the returned struct array. One of the following values: <ul style="list-style-type: none"> 1 means oldest-to-newest (default) -1 means newest-to-oldest
MaxItems	integer	100	Returns up to the specified number of data bars (if available). -1 means all available.
BeginDate	integer or string or datetime object	'1900/01/01' (i.e., from as early as data is available)	<p>Earliest bar that includes a date. Examples:</p> <ul style="list-style-type: none"> 737089 (Matlab datenum format) datetime('Jan 29, 2018') 20180129 (yyyymmdd format) '20180129' '2018/01/29' '2018-01-29' <p>Note: MaxItems has precedence over BeginDate: If there are more data points than MaxItems between BeginDate and yesterday/today⁷³, only the last MaxItems data points (from yesterday/today backward) will be returned, regardless of BeginDate.</p>
Timeout	number	5.0	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols will be done in parallel (see §3.6; Professional IQC license only).
MaxWorkers	integer	(the current parallel pool size, up to 15)	Max number of parallel workers to use (up to the current pool size) when UseParallel =1
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Datestamp, Datenum, High, Low, Open, Close, PeriodVolume, OpenInterest'	<p>Sets the list of data fields reported by IQC for each data bar, as a sub-set of IQFeed's default set of 9 fields.</p> <p>If Fields is set to an empty value ({} or ''), the list of current, available fields is returned.</p> <p>If Fields is not empty, subsequent history queries in the same Matlab session will return the specified fields, in the specified order (Professional IQC license only).</p> <p>Examples:</p> <ul style="list-style-type: none"> 'Datestamp:Open:Close' 'Datestamp,Open,Close' {'Datestamp', 'Open', 'Close'} 'All' (indicates all available fields)

⁷³ Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

5.4 Interval data

To retrieve historic data bars having a custom width, possibly as short as a single second, set **DataType** to 'i' or 'interval', and set the asset's **Symbol**:

```
>> data = IQC('history', 'symbol', 'FB', 'dataType', 'interval')
data =
    100x1 struct array with fields:
        Symbol
        Timestamp
        Datenum
        High
        Low
        Open
        Close
        TotalVolume
        PeriodVolume
        NumberOfTrades

>> data(end)
ans =

        Symbol: 'IBM'
    Timestamp: '2018-03-07 09:43:00'
    Datenum: 737126.404861111
        High: 156.97
        Low: 156.77
        Open: 156.83
        Close: 156.77
    TotalVolume: 215082
    PeriodVolume: 16080
    NumberOfTrades: 0
```

The returned data struct here is similar to the struct returned by the daily, weekly and monthly historical data queries. Unlike those queries, interval-query result does not include an `OpenInterest` field, but does include two new fields: `TotalVolume` (which indicates the total daily volume up to that bar), and `NumberOfTrades`. Also note that we get a `Timestamp` field (US Eastern timezone), not `Datestamp` as with other history queries.

Bars that had no trading action are **not** reported. In the example query above, we see the following `Timestamp` values, where we clearly see a gap during non-trading hours:

```
>> {data.Timestamp}'
ans =
    100x1 cell array
    {'2018-03-06 14:59:00'}
    {'2018-03-06 15:00:00'}
    {'2018-03-06 15:01:00'}
    ... % contiguous data bars
    {'2018-03-06 15:59:00'}
    {'2018-03-06 16:00:00'}
    {'2018-03-06 16:03:00'}
    {'2018-03-06 16:11:00'}
    ...
    {'2018-03-07 08:45:00'}
    {'2018-03-07 09:22:00'}
    {'2018-03-07 09:31:00'}
    {'2018-03-07 09:32:00'}
    ... % contiguous data bars
    {'2018-03-07 09:43:00'}
    {'2018-03-07 09:44:00'}
```

As with the other queries, the current (partial) interval bar is never reported, nor bars that have no data (e.g., 16:04-16:10, 8:34-8:44, 8:46-9:21 in the example above).

The default interval size is 60 secs (aligned on the full-minute mark). You can specify different interval sizes using the **IntervalSize** parameter. For example, a 15-sec interval:

```
>> data=IQC('history','symbol','FB','dataType','interval','intervalSize',15);
```

IQFeed is smart enough to automatically align data bars to full minutes/hours when the requested **IntervalSize** enables this (as is the case for 15 or 60-sec intervals). For example, with 15-sec **IntervalSize** we may get bars for 10:04:30, 10:04:45, 10:05:00. When such alignment is not possible, you will get non-aligned bars. For example, with a 13-sec **IntervalSize**: 09:59:18, 09:59:31, 09:59:57, 10:00:10.

By default, **IntervalSize** specifies the interval's size in seconds and all the bars have this same duration. You can change this by setting the **IntervalType** parameter (default: 'secs') to 'volume' or 'ticks'/'trades'. Naturally, if you change **IntervalType**, the data bars will now have non-equal durations.

```
>> data = IQC('history', 'symbol','FB', 'dataType','interval', ...
              'intervalType','ticks');
```

The **IntervalType** (default: 'secs') and **IntervalSize** (default: 60) parameters should typically be specified together. Note that **IntervalSize** must be a positive integer value (i.e. its value cannot be 4.5 or 0). If **IntervalType** is 'ticks'/'trades', **IntervalSize** must be 2 or higher; If **IntervalType** is 'volume', **IntervalSize** must be 100 or higher; If **IntervalType** is 'secs', **IntervalSize** must be between 1 and 86400 (1 day).⁷⁴

By default, *IQC* reports data in intervals whose labels are set at the *end* of the interval. For example, a data item at 11:12:34 with **IntervalSize**=60 (1 minute) will be included in the interval labeled '11:13:00'. You can modify this default behavior by setting the **LabelAtBeginning** parameter to 1 (or true), so that the labels are set at the *beginning*. In this example, the data item will be reported in the '11:12:00' interval. Note: using **LabelAtBeginning** parameter requires IQFeed client version 6.0 or newer.



By default, *IQC* only reports interval data from today. This means that if you run a query during the weekend, you will not see any data:⁷⁵

```
>> data = IQC('history', 'symbol','FB', 'dataType','interval')
data =
    1×0 empty double row vector
```

You can ask to see additional (older) calendar days by specifying a positive **Days** parameter value. If you set **Days** to -1, then all available information will be reported, subject to the other filter criteria.

Similarly, you can specify a date/time window for the returned data: only bars between the specified **BeginDateTime** and **EndDateTime** (US Eastern time) will be reported, regardless of the value of the **Days** parameter.

Note: queries having **UseParallel**=true are only parallelized if **BeginDateTime** is specified, or if multiple **Symbols** are specified (see below). Single-**Symbol** queries that have an empty (unspecified) **BeginDateTime** are not parallelizable.

⁷⁴ Note that IQFeed's limitations on live 'secs' interval bars (§4.3, §6.3) are stricter than the limitations on historical interval bars: <http://forums.dtn.com/index.cfm?page=topic&topicID=5529>

⁷⁵ *IQC* versions up to 2.42 reported a NO_DATA error in such cases; *IQC* returns [] without an error in version 2.43 or newer.

In addition, you can specify a daily time-window: only bars between **BeginFilterTime** and **EndFilterTime** in each day (US Eastern time-zone) will be reported. This could be useful, for example, to limit the results only to the regular trading hours.

As with the daily bars, you can change the data bars order, using the **DataDirection** parameter (1 means oldest-to-newest (default); -1 means newest-to-oldest).

```
>> data=IQC('history','symbol','FB','dataType','interval','dataDirection',-1);
```

As with the daily bars, you can ask IQFeed to limit the maximal number of data bars (N) using the **MaxItems** parameter:

```
>> data = IQC('history', 'symbol','FB', 'dataType','interval', 'maxItems',20);
```

Note that **MaxItems** takes precedence over **BeginDateTime**, regardless of **DataDirection**. For example, if **MaxItems**=5, you will only get the 5 latest bars (before **EndDateTime**), regardless of the specified **BeginDateTime**.

As with the daily bars, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather data before returning the results (or set to -1 to disable the timeout entirely). This is especially important for historic interval and ticks data queries, since they could return a huge number of data points, which can take a lot of time to download and process.

In addition to **Timeout**, for long queries it is advisable to set the **Progress** parameter to 'console', in order to display a periodic progress update message in the console every 1000 data points (every ~1-2 secs), as well as at the end of the query:

```
>> data = IQC('history', 'symbol','IBM', 'dataType','interval', ...
    'IntervalType','ticks', 'IntervalSize',10, ...
    'BeginDateTime','20200623 100000', ...
    'EndDateTime', '20200623 160000', ...
    'MaxItems',-1, 'timeout',-1, 'progress','console');

1000 history data points processed for IBM. Latest: 2020-06-23 13:30:42 ...
2000 history data points processed for IBM. Latest: 2020-06-23 15:34:06 ...
2460 history data points processed for IBM. Latest: 2020-06-23 15:59:59
```

As with daily data requests, you can request historical data for multiple symbols at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQC('history', 'symbol',{'IBM','GOOG','AAPL'}, ...
    'dataType','interval', 'maxItems',20)

>> data = IQC('history', 'symbol','IBM:GOOG:AAPL', ...
    'dataType','interval', 'maxItems',20) %equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols (3 symbols with 20 data-points each, in this example):

```
data =
20x3 struct array with fields:
    Symbol
    Datestamp
    ...
```

In certain cases, when you request historic data for multiple symbols, you might receive a different number of data bars for different symbols, depending on data availability. In such cases, the result will not be an N-by-M struct array, but a cell array (one cell for each symbol) that contains struct arrays. For example:

```
data =
    1x3 cell array
    {77x1 struct}    {100x1 struct}    {55x1 struct}
```

IQC queries for multiple symbols or a date/time range (i.e., if **BeginDateTime** is specified) can be parallelized using the **UseParallel** parameter, if you have a Professional *IQC* license and Matlab's Parallel Computing Toolbox (see §3.6):

```
>> data = IQC('history', 'dataType','interval', 'UseParallel',true, ...
    'symbol',symbols) % multiple symbols parallelized

>> data = IQC('history', 'dataType','interval', 'UseParallel',true, ...
    'symbol','IBM' ,... % single-symbol date-range parallelized
    'BeginDateTime',20181026100000, ...
    'EndDateTime', 20181026110000)
```

In some cases, users may be tempted to use the historical data mechanism to retrieve real-time data. This is relatively easy to set-up. For example, using an endless Matlab loop that sleeps for 60 seconds, requests the latest historical data for the past minute and then goes to sleep again, or using a periodic timer object that wakes up every minute. In such cases, consider using streaming rather than historical queries (see §6).

Some software vendors make a distinction between intra-day and historical information. However, as far as IQFeed and *IQC* are concerned, this is merely a semantic difference and there is no practical difference.

Note: IQFeed limits interval data to the past 180 calendar days if you make the request outside trading hours, but just past 8 days for requests during US trading hours (9:30-16:30 US Eastern time, Mon-Fri). So, if you request month-old data during trading hours you will get empty results, even if the request was just for a single hour.⁷⁶

The only exception to the 8/180-day limitation are interval bars of full minutes (**IntervalType**='secs' and **IntervalSize** a multiple of 60), since these bars are pre-computed and have a lesser impact on IQFeed's servers. The other interval types are computed on-the-fly from tick data, and so are limited in duration in order not to overload IQFeed's servers, especially during trading hours when server load is high.

IQFeed imposes other limitations based on interval size: minute data is only available since 2005-2007;⁷⁷ longer intervals (daily/weekly/monthly) can access up to 15+ years.⁷⁸

IQFeed subscriptions for daily and intra-day data are different. If you only subscribed for daily data, you will receive a run-time error when fetching intra-day interval data:

```
IQC historic data query (EURGBP.FXCM) error: Unauthorized user ID
(your IQFeed account is not authorized for this data)
```

⁷⁶ The above is true for IQFeed regular accounts; IQFeed trial accounts are limited to only 4 days of intraday data and just one year of daily data (see <https://help.dtniq.com/support-faqs>)

⁷⁷ Specifically for minute (60 sec) intervals, IQFeed's developer FAQ indicates that "Minute interval data dating back to mid 2005 for select contracts and mid 2007 for all others [is available]".

⁷⁸ Again, these values are for regular IQFeed accounts; IQFeed limits trial accounts (see note #76 above)

Also note that IQFeed's interval data typically exclude irregular "O" trades (see §5.5).

By default, *IQC* reports 10 data fields for each interval bar: Symbol, Timestamp, Datenum, High, Low, Open, Close, TotalVolume, PeriodVolume, and NumberOfTrades. If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary). If you have the Professional (or trial) *IQC* license, you can request *IQC* to report fewer data fields, and/or change the reported fields order, using the optional **Fields** parameter. All subsequent interval history queries will report only the requested fields, in the specified order. Fewer fields mean faster processing time and smaller memory usage. Refer to §5.1 for a description of the **Fields** parameter usage.

Finally, note that whereas sub-daily data may report data from non-trading days (e.g., Sunday night, when ES starts trading), these are typically added to the following trading day's bar with daily/weekly/monthly bars.⁷⁹

⁷⁹ <http://forums.dtn.com/index.cfm?page=topic&topicID=5608>

The following parameters affect interval history data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁸⁰	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).
DataDirection	integer	1, meaning oldest bar is first, newest is last	Sets the order of data bars in the returned struct array. One of the following values: <ul style="list-style-type: none"> • 1 means oldest-to-newest (default) • -1 means newest-to-oldest
LabelAtBeginning	logical (true/false)	false	<ul style="list-style-type: none"> • 0: data at 11:17:41 is reported as '11:18' • 1: same data is reported as '11:17'
MaxItems	integer	100	Returns up to the specified number of data bars (if available). -1 means all available.
Days	integer	1 meaning today only	Number of preceding calendar days to process. -1 means unlimited (all available data, subject to the other criteria), 1 means today, 2 means today & yesterday, etc.
IntervalType	string	'secs'	Sets the type of interval size. One of the following values: <ul style="list-style-type: none"> • 's' or 'secs' – time [seconds] (default) • 'v' or 'volume' – traded volume • 't', 'trades' or 'ticks' – number of ticks
IntervalSize	integer	60	Size of bars in IntervalType units. Must be ≥ 1 for secs, ≥ 2 for ticks, ≥ 100 for volume bars
BeginFilterTime	string	'00:00:00'	Only return bars that begin after this time of day (US Eastern time-zone). Only relevant when Days >0 or BeginDateTime is not ". Format: hhmm, hh:mm, hh:mmss or hh:mm:ss
EndFilterTime	string	'23:59:59'	Only return bars that end before this time of day (US Eastern time-zone). Only relevant when Days >0 or BeginDateTime is not ". Format: hhmm, hh:mm, hh:mmss or hh:mm:ss

⁸⁰ In IQC, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
BeginDateTime	integer or string or datetime object	" (empty string) meaning from as early as data is available	Only return bars that begin after this date/time (US Eastern time-zone). Overrides the Days parameter. Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc. Note: MaxItems has precedence over BeginDateTime : If there are more data points than MaxItems between Begin/EndDateTime , only the last MaxItems data points (from EndDateTime backward) are returned, regardless of BeginDateTime .
EndDateTime	integer or string or datetime object	" (empty string) meaning now	Only return bars that end before this date/time (US Eastern time-zone). Overrides the Days parameter. Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc.
Timeout	number	5.0	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If true or 1, and Parallel Computing Toolbox is installed, then querying multiple symbols or a date/time range will be done in parallel (see §3.6; Professional <i>IQC</i> license only).
MaxWorkers	integer	(the current parallel pool size, up to 15)	Maximal number of parallel workers to use (up to the current pool size) when UseParallel =true
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Timestamp, Datenum, High, Low, Open, Close, TotalVolume, PeriodVolume, NumberOfTrades'	Sets the list of data fields reported by <i>IQC</i> for each data bar, as a sub-set of IQFeed's default set of 10 fields. If Fields is set to an empty value ({} or ""), the list of current, available fields is returned. If Fields is not empty, subsequent history queries in the same Matlab session will return the specified fields, in the specified order (Professional <i>IQC</i> license only). Examples: <ul style="list-style-type: none"> • 'Timestamp:Open:Close' • 'Timestamp,Open,Close' • {'Timestamp', 'Open', 'Close'} • 'All' (indicates all available fields)
Progress	string	" (empty string)	When Progress is set to 'console', the download progress is displayed in the console.

5.5 Tick data

Unlike data bars, which aggregate ticks and provide summary information, it is also possible to retrieve historic individual trades (“ticks”). To retrieve this data, set **DataType** to 't' or 'ticks', and set the asset’s **Symbol**:

```
>> data = IQC('history', 'symbol', 'AAPL', 'dataType', 'ticks')
data =
100x1 struct array with fields:
    Symbol
    Timestamp
    Datenum
    Last
    LastSize
    TotalVolume
    Bid
    Ask
    TickID
    BasisForLast
    TradeMarketCenter
    TradeConditions
    TradeAggressorCode
    DayOfMonth
    BasisDescription
    TradeMarketName
    TradeDescription
    AggressorDescription

>> data(end)
ans =
        Symbol: 'AAPL'
    Timestamp: '2019-10-04 09:45:03.862626'
      Datenum: 737702.406294699
         Last: 224.67
    LastSize: 100
  TotalVolume: 5226196
         Bid: 224.66
         Ask: 224.68
        TickID: 7432
   BasisForLast: 'C'
TradeMarketCenter: 19
   TradeConditions: '01'
TradeAggressorCode: 0
      DayOfMonth: 4
BasisDescription: 'Last qualified trade'
TradeMarketName: 'Nasdaq Trade Reporting Facility (NTRF)'
TradeDescription: 'Normal Trade'
AggressorDescription: 'Unknown/unsupported'
```

The data struct here is quite different than the historical bar queries above. Notice the `Timestamp` field, specified in micro-second precision (US Eastern time-zone). See a discussion of the time resolution in the next page. The `DayOfMonth`, `TradeAggressorCode` and `AggressorDescription` fields only appear if you use IQFeed client 6.1 or newer.

Note that the textual Description fields depend on the **MsgParsingLevel** parameter having a value of 2 or higher (see §3.2 and §8)

Also note that only trade ticks are provided, along with the Bid and Ask prices at the time of the trade. IQFeed does not report historic non-trading ticks (i.e., Bid/Ask changes that occurred between the trades).

The `Last` and `LastSize` fields typically refer to the last trade. The type (“basis”) of data in these fields is determined according to the `BasisForLast` field, which is explained in the `BasisDescription` field for convenience.⁸¹ Possible basis values are:⁸²

- C – Last qualified trade.
- E – Extended trade = form T trade.
- O – Other trade = any trade not accounted for by C or E.
- S – Settle = daily settle (only applicable to commodities).

In general, algo-trading should rely only on “C” trades, and potentially also “E” trades. “O” trades often have wide price swings (i.e. large variation from mainstream trading prices); this adds noise to charts and may confuse data analytics.⁸³ IQFeed’s interval data (§5.4) typically exclude such irregular “O” trades.

Note that `TickID` values are not always increasing, and almost never contiguous. They are generally provided by the exchange as unique trade identifiers and so should not be used as an indicator of missing data, and their order is not quarantined. Instead, it is better to rely on the `Timestamp` or `Datum` fields.

In some cases, implied (rather than normal trade) ticks are reported. For example, the following tick was retrieved for the VIX index continuous future (@VX#):

```
>> data = IQC('history', 'symbol', '@VX#', 'dataType', 'ticks');
>> data(1)
ans =
    Symbol: '@VX#'
    Timestamp: '2019-10-04 09:42:41.499000'
    Datum: 737702.404646979
    Last: 18.68
    LastSize: 1
    TotalVolume: 16711
    Bid: 18.65
    Ask: 18.7
    TickID: 6118279
    BasisForLast: 'O'
    TradeMarketCenter: 32
    TradeConditions: '4D'
    TradeAggressorCode: 0
    DayOfMonth: 4
    BasisDescription: 'Other trade = any trade not accounted for by C or E'
    TradeMarketName: 'CBOE Futures Exchange (CFE)'
    TradeDescription: 'Implied'
    AggressorDescription: 'Unknown/unsupported'
```

Note that in the case of @VX# on CBOE, the ticks are only reported in millisecond resolution, not microseconds as for IBM. In this case, `Timestamp` still shows 6 digits after the seconds decimal, but they always end in 000 (...:57.899000). The actual time resolution of reported ticks depends on the specific exchange and security type.⁸⁴

⁸¹ Note that the textual Description fields depend on the `MsgParsingLevel` parameter having a value of 2 or higher (see §3.2)

⁸² Other tick types (basis codes) are NOT reported by IQFeed; <http://forums.dtn.com/index.cfm?page=topic&topicID=5783> (these other tick types are usually rare). Additional tick types may possibly be added by IQFeed in the future.

⁸³ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=3898>

⁸⁴ Micro-second resolution is only available with IQFeed client 5.2 or newer, and only in certain setups (e.g. CMEGroup and equity markets). Contact IQFeed support if you are unsure about the resolution provided by a certain setup configuration.



By default, *IQC* only reports ticks data from today. This means that if you run a query during the weekend, you will not see any data:⁸⁵

```
>> data = IQC('history', 'symbol','FB', 'dataType','ticks')
data =
    1×0 empty double row vector
```

You can ask to see additional (older) calendar days by specifying a positive **Days** parameter value. If you set **Days** to -1, then all available information will be reported, subject to the other filter criteria.

Similarly, you can specify a date/time window for the returned data: only bars between the specified **BeginDateTime** and **EndDateTime** (both of them US Eastern time-zone) will be reported, regardless of the value of the **Days** parameter.

Note: queries having **UseParallel**=true are only parallelized if **BeginDateTime** is specified, or if multiple **Symbols** are specified (see below). Single-**Symbol** queries that have an empty (unspecified) **BeginDateTime** are not parallelizable.

In addition, you can specify a daily time-window: only ticks between **BeginFilterTime** and **EndFilterTime** in each day (US Eastern time-zone) will be reported. This could be useful, for example, to limit the results only to the regular trading hours.

You can also limit the maximal number of ticks using the **MaxItems** parameter.

Note: by default IQFeed limits ticks data to the past 180 calendar days if you make the request outside trading hours, but just past 8 days for requests during US trading hours (9:30-16:30 US Eastern time)⁸⁶ This means that if during trading hours you request historic data from a month ago, you will get none (empty results), even if the request was just for a single hour of data.⁸⁷

You can change the order of the reported ticks, using the **DataDirection** parameter (1 means oldest-to-newest (default); -1 means newest-to-oldest). **MaxItems** has precedence over **BeginDateTime**, regardless of **DataDirection**. For example, if **MaxItems**=5, we'll only get the 5 latest ticks (before **EndDateTime**), regardless of **BeginDateTime**.

As with daily data requests, you can request data for multiple symbols at the same time, in a single *IQC* command by specifying a colon-delimited or cell-array list of **Symbols**:

```
>> data = IQC('history', 'symbol',{'IBM','GOOG','AAPL'}, ...
    'dataType','ticks', 'maxItems',20)

>> data = IQC('history', 'symbol','IBM:GOOG:AAPL', ...
    'dataType','ticks', 'maxItems',20) %equivalent
```

The result will be an array of Matlab structs that correspond to the requested symbols (3 symbols with 20 data-points each, in this example):

```
data =
    20×3 struct array with fields:
        Symbol
        Datestamp
        ...
```

⁸⁵ *IQC* versions up to 2.42 reported a NO_DATA error in such cases; *IQC* returns [] without an error in version 2.43 or newer.

⁸⁶ Historic ticks older than 180 days can be purchased from DTN – <http://forums.iqfeed.net/index.cfm?page=topic&topicID=4376>

⁸⁷ This is true for regular IQFeed accounts; IQFeed trials are limited to 4 days of intraday data (<https://help.dtniq.com/support-faqs>)

In certain cases, when you request historic data for multiple symbols, you might receive a different number of data bars for different symbols, depending on data availability. In such cases, the result will not be an N-by-M struct array, but a cell array (one cell for each symbol) that contains struct arrays. For example:

```
data =
    1x3 cell array
    {77x1 struct}    {100x1 struct}    {55x1 struct}
```

IQC queries for multiple symbols or a date/time range (i.e., if **BeginDateTime** is specified) can be parallelized using the **UseParallel** parameter, if you have a Professional *IQC* license and Matlab's Parallel Computing Toolbox (see §3.6):

```
>> data = IQC('history', 'dataType','ticks', 'UseParallel',true, ...
    'symbol',symbols) % multiple symbols parallelized

>> data = IQC('history', 'dataType','ticks', 'UseParallel',true, ...
    'symbol','IBM' ,... % single-symbol date-range parallelized
    'BeginDateTime',20181026100000, ...
    'EndTime', 20181026110000)
```

By default, *IQC* reports 18 data fields for each tick: Symbol, Timestamp, Datenum, Last, LastSize, TotalVolume, Bid, Ask, TickID, BasisForLast, TradeMarketCenter, TradeConditions, TradeAggressorCode, DayOfMonth, BasisDescription, TradeMarketName, TradeDescription, and AggressorDescription.⁸⁸ If the **Fields** parameter is set to an empty value ({} or ""), the current set of fields and the full list of available fields, are reported (in this case, a **Symbol** parameter is unnecessary). If you have the Professional (or trial) *IQC* license, you can request *IQC* to report fewer data fields, and/or change the reported fields order, using the optional **Fields** parameter. All subsequent tick history queries will report only the requested fields, in the specified order. Fewer fields mean faster processing time and smaller memory usage. Refer to §5.1 for a description of the **Fields** parameter usage.

Finally, as with other *IQC* commands, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather data before returning the results (or set to -1 to disable the timeout entirely). This is especially important for historic interval and ticks data queries, since they could return a huge number of data points, which can take a lot of time to download and process.

In addition to **Timeout**, for long queries it is advisable to set the **Progress** parameter to 'console', in order to display a periodic progress update message in the console every 1000 data points (every ~1-2 secs), as well as at the end of the query:

```
>> data = IQC('history', 'symbol','IBM', 'dataType','ticks', ...
    'BeginDateTime','20200623 150000', ...
    'EndTime', '20200623 152000', ...
    'MaxItems',-1, 'timeout',-1, 'progress','console');

1000 history data points processed for IBM. Latest: 2020-06-23 15:05:06.665127...
2000 history data points processed for IBM. Latest: 2020-06-23 15:11:43.512718...
3000 history data points processed for IBM. Latest: 2020-06-23 15:16:47.954088...
3489 history data points processed for IBM. Latest: 2020-06-23 15:20:00.076970
```

⁸⁸ The textual fieldsBasisDescription,TradeMarketName,TradeDescriptionandAgressorDescriptionare *IQC*-generated textual interpretations of the codes in the IQFeed-generatedBasisForLast,TradeMarketCenter,TradeConditionsandTradeAggressorCodefields respectively,as governed by the **MsgParsingLevel**parameter (see §3.2)

The following parameters affect ticks history data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ⁸⁹	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s). Multiple symbols can be parallelized using the UseParallel parameter (see below).
DataDirection	integer	1 meaning oldest tick is first, newest last	Sets the order of ticks in the returned struct array. One of the following values: <ul style="list-style-type: none"> • 1 means oldest-to-newest (default) • -1 means newest-to-oldest
MaxItems	integer	100	Returns up to the specified number of ticks (if available). -1 means all available.
Days	integer	1 meaning today only	Number of preceding calendar days to process. -1 means unlimited (all available data, subject to the other criteria), 1 means today, 2 means today & yesterday, etc.
BeginFilterTime	string	'00:00:00'	Only return ticks that begin after this time of day (US Eastern). Only relevant when Days >0 or BeginDateTime is not ". Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
EndFilterTime	string	'23:59:59'	Only return ticks that end before this time of day (US Eastern). Only relevant when Days >0 or BeginDateTime is not ". Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
BeginDateTime	integer or string or datetime object	" (empty string) meaning from as early as data is available	Only return ticks that begin after this date/time (US Eastern time-zone). Overrides the Days parameter. Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc. Note: MaxItems has precedence over BeginDateTime : If there are more data points than MaxItems between BeginDateTime and EndDateTime , only the last MaxItems data points (from EndDateTime backward) will be returned, regardless of BeginDateTime .

⁸⁹ In IQC, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
EndDateTime	integer or string or datetime object	" (empty string) meaning now	Only return ticks that end before this date/time (US Eastern time-zone) Overrides the Days parameter. Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc.
Timeout	number	5.0	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying multiple symbols or a date/time range will be done in parallel (see §3.6; Professional <i>IQC</i> license only).
MaxWorkers	integer	(the current parallel pool size, up to 15)	Maximal number of parallel workers to use (up to the current pool size) when UseParallel =true
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Timestamp, Datenum, Last, LastSize, TotalVolume, Bid, Ask, TickID, BasisForLast, TradeMarketCenter, TradeConditions, TradeAggressorCode, DayOfMonth, BasisDescription, TradeMarketName, TradeDescription, AggressorDescription'	Sets the list of data fields reported by <i>IQC</i> for each data bar, as a sub-set of IQFeed's default set of 18 fields. If Fields is set to an empty value ({} or ""), the list of current, available fields is returned. If Fields is not empty, subsequent history queries in the same Matlab session will return the specified fields, in the specified order (Professional <i>IQC</i> license only). Examples: <ul style="list-style-type: none"> • 'Timestamp:Bid:Ask' • 'Timestamp,Bid,Ask ' • {'Timestamp', 'Bid', 'Ask'} • 'All' (indicates all available fields)
Progress	string	" (empty string)	When Progress is set to 'console', the data download progress is periodically displayed in the console.

5.6 Market summary data and scanner

All the queries described so far in this chapter return historic data for individually-specified **Symbols**. We can retrieve historic end-of-day market state (quotes/trades and fundamental data) of all traded securities as-of a single historic date (May 20, 2018 or later), using a 'summary' query (see §4.6) with a non-default **Date** parameter:

```
>> data = IQC('summary', 'Date', 20190110) %all NYSE equities on Jan 10, 2019
data =
4706x1 struct array with fields:
    Symbol
    Exchange
    Type
    Last
    ...      (total of 28 data fields)

>> data(1)
ans =
struct with fields:
    Symbol: 'A'
    Exchange: 7
    Type: 1
    Last: 69.9
    TradeSize: 3350
    TradedMarket: 7
    TradeDate: 20190110
    TradeTime: 180131
    Open: 69.05
    High: 69.95
    Low: 68.6
    Close: 69.25
    Bid: 50
    BidMarket: 11
    BidSize: 400
    Ask: 69.9
    AskMarket: 11
    AskSize: 100
    Volume: 1080882
    PDayVolume: 2442291
    UpVolume: 413506
    DownVolume: 231604
    NeutralVolume: 435772
    TradeCount: 10340
    UpTrades: 3070
    DownTrades: 2819
    NeutralTrades: 4447
    VWAP: 69.5782
```

This query shows that 4706 equities were traded on NYSE on Jan 10, 2019. The data may change over time, as DTN retroactively fixes its historic records.

The default **DataType** parameter value ('snapshot') fetches end-of-day trading data. To fetch end-of-day fundamental data, set **DataType**='fundamental'.⁹⁰

```
>> data = IQC('summary', 'Date', 20190110, 'DataType', 'fundamental');
```

Note that there is no **Symbol** parameter in a 'summary' query – data for all the symbols that match the specified **SecType** (default: 'equity'), **Exchange** (default: 'NYSE') and/or **Date** (default: now/latest) is returned. For historic snapshot trading data of specific symbols, use one of the other query types (§5.1-§5.5). Unfortunately, there is no corresponding alternative for historic fundamental data of specific symbols.

⁹⁰ Note that we only receive 4705 securities in the fundamental query compared to 4706 securities for the snapshot query (ASXw has snapshot data but no fundamentals)– this is a data error. With NYSE bonds on the same date we see a similar phenomenon: three symbols (CVS.24.CB, DUK.46B.CB, TXT27.CB) have snapshot data but no fundamentals. All of these are data errors.

We can filter the returned data for various criteria using the **Filter** parameter (see §4.6), effectively serving as a market scanner for the requested historic date.

Using end-of-day historic summary query enables fetching the data for expired contracts and delisted securities, which are no longer traded. Fetching historic data for such non-trading symbols using any other query type is not possible.⁹¹

Note: Market summaries are only available with IQFeed client 6.1 or newer, and only if you are subscribed to the requested data at DTN and IQFeed has relevant history data (data is only available for trading days since May 20, 2018, and only for some **SecType**/**Exchange** combinations). Otherwise, you may receive an error such as one of these:

```
The 'summary' query is only supported by IQFeed client 6.1 or newer; you are
using version 6.0.
IQFeed market summary query error: Code: 50004 - User not authorized for
market summary file requested.
IQFeed market summary query error: Code: 50007 - No file available for NASDAQ
on 2020-11-29 (possibly a non-trading day; try a different Date).
```

The following parameters affect historic market summary queries (see §4.6 for details):

Parameter	Data type	Default	Description
DataType	string	'snapshot'	Either 'snapshot' or 'fundamental' (not 'top')
Exchange	string	'NYSE'	One of the markets listed in §8.3
SecType	string	'Equity'	One of the security types listed in §8.4
Date	integer or string or datetime object	now (latest available data)	Date for which to fetch the end-of-day data. Examples: <ul style="list-style-type: none"> 737089 (Matlab datenum format) datetime('Jan 29, 2018') 20180129 (yyyymmdd format) '20180129' '2018/01/29' '2018-01-29'
ReportEmptyFields	logical	false or 0	If true, then irrelevant data fields (which contain empty [] values for all securities) are reported; if false (default), they are not
Filter	string or cell-array of strings	{}	Zero or more filter criteria (condition strings) – Matlab expression(s) involving the reported data fields, which result in a logical (true/false) value. Examples: <ul style="list-style-type: none"> 'MaturityDate > 20241231' 'MarketCap > 5000 & PeRatio < 9' {'MarketCap > 5000', 'Beta >= 1.2'}
Timeout	number	300	Max number of seconds to wait for incoming data (0-9000, where 0 means infinite)



Note: market summary functionality is only available in the Professional *IQC* license

⁹¹ A [huge] static text file containing a [very long] list of expired symbols is available for download from DTN's FTP site (<http://www.dtniq.com/beta/IEOPTION.zip>; see <http://forums.iqfeed.net/index.cfm?page=topic&topicID=3326> for details). Note that this file is not actively maintained, so it is better to use the API functionality via *IQC*.

6 Streaming data

Streaming data is a near-real-time mechanism, where IQFeed sends ongoing asynchronous update messages to *IQC* of tick (quote and trade) and news events.

These events are accumulated in *IQC* memory buffers. They can either be queried **asynchronously** (via ad-hoc queries that “peek” at the latest accumulated data without disturbing the ongoing streaming, as shown in §6.1-§6.4), or handled **synchronously** (immediately as each event is received, using callbacks (§10) or alerts (§11)).

Depending on your IQFeed subscription, streaming may be delayed by 10+ minutes compared to a real-time feed (a real-time data subscription is needed for live data).⁹²

6.1 Streaming quotes

The streaming quotes mechanism has two distinct parts:

1. Request IQFeed to start sending a stream of quotes for a specified security. This is done by using the 'quotes' action and setting a **NumOfEvents** parameter to a positive >1 value.
2. At any later time(s), you can access the accumulated quotes using either of the following alternatives:
 - a. Use 'quotes' action and **NumOfEvents** of -1 (minus one). This will return the latest streamed data, without stopping the background streaming.
 - b. If you set the **AssignTo** variable in the original request, you can directly access the streamed data in the specified base workspace variable.

For example, to request 100 streaming quotes for a continuous VIX future contract:

```
IQC('quotes', 'Symbol','@VX#', 'NumOfEvents',100)
```

IQFeed will start sending quotes to *IQC* in the background, up to the specified **NumOfEvents**, without affecting normal Matlab processing. You can continue to work in Matlab, process/display information etc., while quotes accumulate in the background.



Quotes will only stream in the background in non-blocking mode. If you assign the *IQC* command results to a variable, the request is treated as blocking and *IQC* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

```
IQC('quotes', 'Symbol','@VX#', 'NumOfEvents',100); % streaming, non-blocking
data = IQC('quotes', 'Symbol','@VX#', 'NumOfEvents',100); % blocking
```

NumOfEvents can be any number >1 for streaming (a value of 1 is the standard snapshot query in §4.1). To collect streaming quotes endlessly, set **NumOfEvents** to `inf`. Note that in Matlab, `inf` is a number (not a string), so do not enclose it in quotes (`'inf'`).

The quotes are collected into an internal data buffer in *IQC*. A different buffer is maintained for each symbol. The buffer size can be controlled using the **MaxItems** parameter, which has a default value of 1. This means that by default only the latest streaming quote of each type (bid/ask) is stored, along with high/low/close data.

If you set a higher value for **MaxItems**,⁹³ then up to the specified number of latest quotes will be stored. For example, to store the latest 5 quotes:

```
IQC('quotes', 'Symbol','@VX#', 'NumOfEvents',100, 'MaxItems',5)
```

⁹² See §9.3 for a programmatic method to determine whether your exchange subscription is delayed or real-time.

⁹³ **MaxItems** is a numeric parameter like **NumOfEvents**, so don't enclose the parameter value within string quotes (``')



Note: **MaxItems** increases memory usage, multiplied by the number of streamed symbols⁹⁴

Subsequent requests to retrieve the latest accumulated quotes buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one). These requests return a Matlab data struct similar to the following:

```
>> data = IQC('quotes', 'Symbol', '@VX#', 'NumOfEvents', -1)
data =
    Symbol: '@VX#'
    Command: 'w@VX#'
    isActive: 1
    EventsToProcess: 100
    EventsProcessed: 57
    LatestEventDatetime: 737128.637260451
    LatestEventTimestamp: '20180309 15:17:39'
    DataType: 'quotes and trades'
    ProcessType: 'stream'
    AssignTo: ''
    errorMsg: ''
    BufferSize: 5
    Buffer: [5x1 struct]
    LatestData: [1x1 struct]
```

In the returned data struct, we can see the following fields:

- **Symbol** – the requested Symbol.
- **Command** – the command sent to IQFeed, including the requested Symbol.
- **isActive** – logical flag indicating whether quotes are currently streamed for this security. When **NumOfEvents** ticks are received, this flag is set to false (0).
- **EventsToProcess** – total number of streaming ticks requested for the security (using the **NumOfEvents** parameter).
- **EventsProcessed** – number of streaming ticks received for this security. When **EventsProcessed** >= **EventsToProcess**, streaming quotes are turned off and **isActive** is set to false (0). Note that it is possible that **EventsProcessed** > **EventsToProcess**, since it takes a while for the streaming cancellation request to reach IQFeed, and during this time a few additional ticks may have arrived.
- **LatestEventDatetime** – Matlab numeric datetime representation of the **LatestEventTimestamp** (local timezone).
- **LatestEventTimestamp** – timestamp (string format) when this quote event was received by IQC (local timezone).
- **DataType** – type of data to stream (set by **DataType** parameter, see below).
- **ProcessType** – always equal to 'stream' for streaming quotes.
- **AssignTo** – the name of the assigned variable in the base Matlab workspace (= **AssignTo** parameter, see below). In normal cases, this field is empty ('').
- **errorMsg** – contains the error message in case streaming cannot be processed (for example: 'Symbol not found'). In normal cases, this field is empty ('').
- **BufferSize** – size of the data buffer (= **MaxItems** parameter, see below).
- **Buffer** – buffer of size **BufferSize**, accumulating the latest quote updates.
- **LatestData** – latest quote event received from IQFeed.

⁹⁴ Quotes use up to ~16KB of Matlab memory (depending on the selected **Fields**). So, if **MaxItems**=1000, each streamed symbol would need up to 1000*16KB = 16MB of Matlab memory when its buffer becomes full (which could take a while). Streaming 100 symbols will require up to ~1.6GB. Such large memory usage might significantly degrade overall performance.

Here is a simulated timeline that illustrates the use of streaming data in *IQC*:

Time	Events so far	User command	Description
9:50:00	0	<code>IQC('quotes', 'Symbol','IBM', 'MaxItems',100, 'NumOfEvents',100)</code>	Streaming data for IBM starts. Up to 100 events to accumulate.
9:50:10	23	<code>data = IQC('quotes', 'symbol','IBM', 'NumOfEvents',-1)</code>	Return the 23 accumulated quotes; background streaming continues.
9:50:20	42	<code>data = IQC('quotes', 'symbol','IBM', 'NumOfEvents',-1)</code>	Return the 42 accumulated quotes; background streaming continues.
9:50:30	57	<code>IQC('quotes', 'Symbol','IBM', 'MaxItems',80, 'NumOfEvents',80)</code>	Reduce max # of events 100→80. Only 57 events accumulated until now, so streaming continues.
9:50:40	65	<code>data = IQC('quotes', 'symbol','IBM', 'NumOfEvents',-1)</code>	Return the 65 accumulated quotes; background streaming continues.
9:50:50	72	<code>IQC('quotes', 'Symbol','IBM', 'NumOfEvents',0)</code>	Reduce max # of events 80→0. 72 events accumulated until now, so streaming stops immediately.

Different quotes are sent independently from IQFeed server with a unique timestamp.

Note: `data.LatestEventDatetime` and `data.LatestEventTimestamp` are specified in local time-zone. In contrast, `data.LatestData.Most_Recent_Trade_Time` and `data.Buffer.-Most_Recent_Trade_Time` use the server time-zone, typically US Eastern.

To get the quotes data, simply read the fields of the returned `data` struct, for example:⁹⁵

```
>> data.LatestData
ans =

                Symbol: '@VX#'
      Most_Recent_Trade: 17.08
  Most_Recent_Trade_Size: []
  Most_Recent_Trade_Time: '08:06:20.716000'
Most_Recent_Trade_Market_Center: 32
      Total_Volume: 4507
                Bid: 17.05
      Bid_Size: 63
                Ask: 17.1
      Ask_Size: 244
                Open: 17.2
                High: 17.35
                Low: 17
                Close: 17.23
  Message_Contents: 'Cbasohlcv'
  Message_Description: 'Last qualified trade; A bid update
                        occurred; An ask update occurred; A
                        settlement occurred; An open declaration
                        occurred; A high declaration occurred; A
                        low declaration occurred; A close
                        declaration occurred; A volume update
                        occurred'
  Most_Recent_Trade_Conditions: '4D'
  Trade_Conditions_Description: 'Implied'
  Most_Recent_Market_Name: 'CBOE Futures Exchange (CFE)'
```

⁹⁵ The textual description fields depend on the **MsgParsingLevel** parameter having a value of 2 or higher (see §3.2 and §8)

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.⁹⁶

Each streaming security asset can have a different `BufferSize`, by specifying a different **MaxItems** value in the command (large for heavily-traded assets, small for others).

Once the data is retrieved, you can direct *IQC* to clear (empty) the internal `Buffer`, by setting **ClearBuffer** to true or 1. The latest buffer will be returned, and the internal `Buffer` (but no other field) will be immediately emptied, awaiting new streaming quotes:⁹⁷

```
data = IQC('quotes', 'symbol', 'IBM', 'NumOfEvents', -1, 'ClearBuffer', true);
```

To stop collecting streaming quotes, simply resend a request with **NumOfEvents**=0:

```
IQC('quotes', 'symbol', 'IBM', 'NumOfEvents', 0);
```

IQFeed reports 16 standard data fields by default. If you have the Professional (or trial) *IQC* license, you can customize the returned data fields by requesting up to 50+ additional fields, removing standard fields, and setting the order of the reported fields. This can be done using the **Fields** parameter, as explained in §4.1. For example:

```
IQC('quotes', 'symbol', 'IBM', 'fields', 'Last,Ask,Bid', 'numOfEvents', 6);
```

When **DataType** is 'q' or 'quotes', whenever any of the requested data fields (either the standard 16 fields, or a customized set) gets updated (not necessarily to a different value), a new tick (update/quote) message is sent/streamed. Adding data fields means a corresponding increase in tick messages. It is not possible in IQFeed to request data fields without the corresponding update messages for these fields (or vice versa). The only exception to this rule is setting **DataType** to 't' or 'trades': in this case only trade updates (containing all the requested fields) will be streamed, but no field updates.

In summary, the fewer data fields that are requested, the faster the run-time processing, and the lower the corresponding tick message rate, thus enabling a larger number of usable quotes to be streamed and processed by your Matlab program each second.

You can specify multiple symbols for streaming at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
IQC('quotes', 'symbols', {'IBM', 'GOOG', 'AAPL'}, 'numOfEvents', 6);
IQC('quotes', 'symbols', 'IBM:GOOG:AAPL', 'numOfEvents', 6); % equivalent
```

And similarly, when retrieving the accumulated streaming data:

```
>> data = IQC('quotes', 'symbol', 'IBM:GOOG:AAPL', 'numOfEvents', -1);
data =
    1x3 struct array with fields:
        Symbol
        Command
        isActive
        EventsToProcess
        EventsProcessed
        LatestEventDatetime
        LatestEventTimestamp
        DataType
        ProcessType
        BufferSize
        Buffer
        LatestData
```

⁹⁶ When **NumOfEvents** events have been received, IQFeed is instructed to stop streaming updates, but some update messages may already be on their way from IQFeed before streaming actually stops. These extra update messages are not accumulated in the `Buffer`, but the latest of these messages will be reflected in `LatestData` field.

⁹⁷ During the buffer clear operation some streaming data may be lost, so it is advised not to clear too often.

```
>> data(1).LatestData
ans =
    struct with fields:
        Symbol: 'IBM'
        Most_Recent_Trade: 142.48
        Most_Recent_Trade_Size: 41149
        Most_Recent_Trade_Time: '17:33:40.531781'
        Most_Recent_Trade_Market_Center: 19
        ...
```

To get the latest data for all streamed symbols, omit the **Symbol** parameter (or set it to empty[""]) in the *IQC* command. Note: this will return both active and non-active streams:

```
>> data = IQC('quotes', 'numOfEvents', -1); % no symbol: return ALL streams
data =
    1x5 struct array with fields:
        Symbol
        Command
        ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to "").⁹⁸

```
>> IQC('quotes', 'numOfEvents', 0); % no symbol: stop ALL streams
```

IQFeed typically allows streaming up to 500 symbols. This limit can be increased by paying DTN for increased data subscriptions. In any case, the actual maximal number of concurrently-streaming symbols is limited by performance considerations (see §3.6).

Note that during non-trading hours, there is no streaming data (of course). To test the streaming-data mechanism during non-trading hours, use the dummy symbol TST\$Y, for which IQFeed sends a continuous 24/7 stream of pre-recorded data.⁹⁹

When debugging streaming, it is sometimes useful to assign the streamed data to a base Matlab workspace variable, which can be monitored live using Matlab's Variable Explorer or directly checked/used within a Matlab script, without a need to call `IQC('quotes', ..., 'numOfEvents', -1)` to constantly refetch the latest data. This data mirroring can be done by setting the **AssignTo** parameter to any valid Matlab variable name or assignment target, for example: 'myData' or 'streamed.VX' or 'data{3}'.

```
>> IQC('quotes', ..., 'AssignTo', 'myData');
% streaming data in background is mirrored in base workspace variable myData:
>> myData
myData =
    struct with fields:
        Symbol: '@VX#'
        Command: 'BW,@VX#,60,20200421 000000,7,100,,,B,s,,5'
        isActive: 1
        EventsToProcess: Inf
        EventsProcessed: 129
        LatestEventDatetime: 737902.831878183
        LatestEventTimestamp: '20200421 19:57:54'
        DataType: 'quotes and trades'
        ProcessType: 'stream'
        AssignTo: 'myData'
        errorMsg: ''
        MaxDaysToProcess: 7
        BufferSize: 100
        Buffer: [100x1 struct]
        LatestData: [1x1 struct]
```

⁹⁸ Note that cancelling all active streams cancels streaming regional updates (§6.2) in addition to streaming quotes.

⁹⁹ IQFeed's streaming functionality of TST\$Y is currently broken. To get notified when DTN reports that the functionality is fixed, follow this forum thread: <http://forums.dtn.com/index.cfm?page=topic&topicID=4286>

Here is a summary of the *IQC* parameters that directly affect streaming quotes:

Parameter	Data type	Default	Description
Symbol or Symbols ¹⁰⁰	colon or comma-delimited string, or cell-array of strings	(none)	Limits request to specified symbol(s). E.g.: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0
NumOfEvents	integer	MaxItems	One of: <ul style="list-style-type: none"> • inf – continuous endless streaming quotes for the specified security • N>1 – stream only N quotes • 1 – get only a single quote (default) • 0 – stop streaming quotes • -1 – return latest accumulated quotes data while continuing to stream new quotes data
MaxItems	integer	1	Number of streaming quotes stored in cyclic buffer. Once this number of quotes are received old quotes are discarded as new quotes arrive.
DataType	string	'q'	One of: <ul style="list-style-type: none"> • 'q' or 'quotes' (default) – stream both trades & quote (bid/ask update) events • 't' or 'trades' – stream trade events only
Fields	colon or comma-separated string, or cell-array of strings	'Symbol, Most Recent Trade, Most Recent Trade Size, Most Recent Trade Time, ...' (see §4.1)	Sets the list of data fields reported by IQFeed for each quote. IQFeed's default set has 16 fields; 50+ additional fields can be specified. If Fields is set to an empty value ({} or ""), the list of current, available fields is returned. If Fields is not empty, subsequent quotes queries will return the specified fields, in the specified order (Professional <i>IQC</i> license only). The Symbol field is always returned first, even if not specified. Examples: <ul style="list-style-type: none"> • {'Bid', 'Ask', 'Last'} • 'Bid, Ask, Last' • 'Bid:Ask:Last' • 'All' (indicates all available fields)
ClearBuffer	logical (true/false)	false	If true or 1, the internal cyclic quotes buffer is cleared after the data is returned to the caller.
AssignTo	string	" (empty string)	Contains the assignment target (typically a variable name) in the base Matlab workspace, useful for stream debugging.

¹⁰⁰ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

6.2 Regional updates

Regional quotes are Bid and Ask prices delivered from various regional markets (exchanges). The streaming regional market update mechanism has two parts, just like streaming ticks (§6.1):

1. Request IQFeed to start sending a stream of regional updates. This is done by using the 'regional' action and setting a **NumOfEvents** parameter to a positive >1 value. You must specify the **Symbol(s)** for which regional updates will stream.
2. At any later time(s), you can access the accumulated regional updates using either of the following alternatives:
 - a. Use 'regional' action and **NumOfEvents** of -1 (minus one). This will return the latest streamed data, without stopping the background streaming.
 - b. If you set the **AssignTo** variable in the original request, you can directly access the streamed data in the specified base workspace variable.

For example, to request 100 streaming regional updates for Facebook:

```
IQC('regional', 'Symbol','FB', 'NumOfEvents',100)
```

This causes IQFeed to start sending regional updates to *IQC* in the background, up to the specified **NumOfEvents**, without affecting normal Matlab processing. You can continue to work with Matlab, process and display information etc., while the regional updates accumulate in the background.



Regional updates will only stream in the background in non-blocking mode. If you assign the *IQC* command results to a variable, the request is treated as blocking and *IQC* will wait for all data to accumulate (or **Timeout** to occur), as described in §7.2:

```
IQC('regional', 'Symbol','FB', 'NumOfEvents',100); % streaming, non-blocking
data = IQC('regional', 'Symbol','FB', 'NumOfEvents',100); % blocking
```

NumOfEvents can be any number higher than 1 for streaming to work (a value of 1 is the standard snapshot regional-update request described in §7.2). To collect streaming regional updates endlessly, set **NumOfEvents** to the value `inf`. Note that in Matlab, `inf` is a number (not a string), so do not enclose it in quotes ('inf').

The regional updates are collected into an internal data buffer in *IQC*. A different buffer is maintained for each symbol. The buffer size can be controlled using the **MaxItems** parameter, which has a default value of 1¹⁰¹. This means that by default only the latest streaming regional update that affect the specified symbols will be stored in the buffer and become accessible for later processing.

If you set a higher value for **MaxItems**, then up to the specified number of latest regional update items will be stored. For example, to store the latest 5 updates:

```
IQC('regional', 'Symbol','FB', 'NumOfEvents',100, 'MaxItems',5)
```



Note that using a large **MaxItems** increases memory usage. This could have an adverse effect if you set a very large buffer size (many thousands) and/or streaming of a large number of different securities.¹⁰²

¹⁰¹ Note that **MaxItems** is a numeric parameter like **NumOfEvents**, so don't enclose the parameter value within string quotes ('')

¹⁰² Each regional update item uses 2KB of Matlab memory. During trading hours, there could be dozens of updates per second for highly liquid symbols (i.e., 500MB or more per hour, if all updates are saved). Limiting **MaxItems** to some finite value ensures that the memory usage and performance impact remain low.

Subsequent requests to retrieve the latest accumulated regional updates buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one). These requests return a Matlab data struct similar to the following:

```
>> data = IQC('regional', 'Symbol','FB', 'NumOfEvents',-1)
data =
    Symbol: 'FB'
    Command: 'S,REGON,FB'
    isActive: 1
    EventsToProcess: 100
    EventsProcessed: 83
    LatestEventDenum: 737146.784037153
    LatestEventTimestamp: '20180327 18:49:00'
    DataType: 'regional'
    ProcessType: 'stream'
    AssignTo: ''
    errorMsg: ''
    BufferSize: 5
    Buffer: [5x1 struct]
    LatestData: [1x1 struct]
```

In the returned data struct, we can see the following fields:

- **Symbol** – the requested Symbol.
- **Command** – the command sent to IQFeed, including the requested Symbol.
- **isActive** – a logical flag indicating whether regional updates are currently being streamed for this security. When **NumOfEvents** ticks have been received, this flag is set to false (0).
- **EventsToProcess** – total number of streaming regional updates requested (using the **NumOfEvents** parameter).
- **EventsProcessed** – number of streaming regional updates received. When **EventsProcessed** \geq **EventsToProcess**, streaming updates are turned off and **isActive** is set to false (0).

Note that it is possible that **EventsProcessed** $>$ **EventsToProcess**, since it takes a while for the streaming cancellation request to reach IQFeed and during this time a few additional update messages may have arrived.

- **LatestEventDenum** – Matlab numeric denum representation of the **LatestEventTimestamp** (local timezone).
- **LatestEventTimestamp** – timestamp (string format) when this regional update event was received by IQC (local timezone).
- **DataType** – always equal to 'regional' for streaming regional updates.
- **ProcessType** – always equal to 'stream' for streaming regional updates.
- **AssignTo** – the name of the assigned variable in the base Matlab workspace (=AssignTo parameter, see below). In normal cases, this field is empty (").
- **errorMsg** – contains the error message in case streaming cannot be processed (for example: 'Symbol not found'). In normal cases, this field is empty (").
- **BufferSize** – size of the data buffer (=MaxItems parameter, see below).
- **Buffer** – buffer of size **BufferSize**, accumulating the latest regional updates.
- **LatestData** – latest regional update event received from IQFeed.

To get the regional updates data, simply read the fields of the returned data struct:¹⁰³

¹⁰³ The textual Description fields depend on the **MsgParsingLevel** parameter having a value of 2 or higher (see §3.2 and §8)

```
>> data.LatestData
ans =
    RegionalBid: 155.34
    RegionalBidSize: 100
    RegionalBidTime: '12:29:45'
    RegionalAsk: 155.55
    RegionalAskSize: 200
    RegionalAskTime: '12:29:45'
    FractionDisplayCode: 14
    DecimalPrecision: 4
    FractionDisplayDescription: 'Four decimal places'
    MarketCenter: 11
    MarketCenterDescription: 'NYSE Archipelago (NYSE_ARCA)'
```

Each update has an associated timestamp, since different regional updates are sent separately and independently from IQFeed server.

Note: `data.LatestEventDatetime` and `data.LatestEventTimestamp` are specified in the local time-zone; in contrast, `data.LatestData.RegionalBidTime` and `.RegionalAskTime` are specified in the server's time-zone (typically US Eastern time zone).

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.¹⁰⁴

Each streaming security asset can have a different `BufferSize`, by specifying a different **MaxItems** value in the streaming command. This can be used for specifying a larger **MaxItems** for heavily-traded assets vs. lightly-traded ones.

Once the data is retrieved, you can direct *IQC* to clear (empty) the internal `Buffer`, by setting **ClearBuffer** to true or 1. The latest buffer will be returned, and the internal `Buffer` (but no other field) will be immediately emptied, awaiting new regional updates:¹⁰⁵

```
data = IQC('regional', 'symbol', 'FB', 'NumOfEvents', -1, 'ClearBuffer', true);
```

To stop collecting regional updates, simply resend a request with **NumOfEvents**=0:

```
IQC('regional', 'symbol', 'FB', 'NumOfEvents', 0);
```

You can specify multiple symbols for streaming at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
IQC('regional', 'symbols', {'IBM', 'GOOG', 'AAPL'}, 'numOfEvents', 6);
IQC('regional', 'symbols', 'IBM:GOOG:AAPL', 'numOfEvents', 6); % equivalent
```

As with streaming quotes (§6.1), to get the latest data for all streamed symbols, omit the **Symbol** parameter or set it to empty `[""]`. This returns all streams (both active/not):

```
>> data = IQC('regional', 'numOfEvents', -1); % no symbol: get ALL streams
data =
    1x5 struct array with fields:
        Symbol
        Command
        isActive
        EventsToProcess
        ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to `""`):¹⁰⁶

```
>> IQC('regional', 'numOfEvents', 0); % no symbol: ALL streams are stopped
```

¹⁰⁴ When **NumOfEvents** events have been received, IQFeed is instructed to stop streaming updates, but one or more update messages may already be on their way from IQFeed before streaming actually stops. These extra update messages are not accumulated in the `Buffer`, but the latest of these messages will be reflected in `LatestData` field.

¹⁰⁵ During and around the time of the buffer clear, some streaming data may be lost, so it is advised not to clear too often...

¹⁰⁶ Note that cancelling all active streams cancels streaming quotes (§6.1) in addition to streaming regional updates.

When debugging streaming, it is sometimes useful to assign the streamed data to a base Matlab workspace variable, which can be monitored live using Matlab's Variable Explorer or directly checked/used within a Matlab script, without a need to call `IQC('regional',..., 'numOfEvents',-1)` to constantly refetch the latest data. This data mirroring can be done by setting the **AssignTo** parameter to any valid Matlab variable name or assignment target, for example: 'myData' or 'streamed.FB' or 'data{3}'.

```
>> IQC('regional', ..., 'AssignTo','myData');
% streaming data in background is mirrored in base workspace variable myData:
>> myData
myData =
        Symbol: 'FB'
      Command: 'S,REGON,FB'
      isActive: 1
EventsToProcess: 100
EventsProcessed: 83
    ...
```

Here is a summary of the *IQC* parameters that affect streaming regional updates:

Parameter	Data type	Default	Description
Symbol or Symbols ¹⁰⁷	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: <ul style="list-style-type: none"> '@VX#' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM','AAPL','GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0
NumOfEvents	integer	MaxItems	One of: <ul style="list-style-type: none"> inf – continuous endless streaming regional updates for specified security N>1 – stream only N regional updates 1 – get only a single update (default) 0 – stop streaming regional updates -1 – return the latest accumulated regional updates data while continuing to stream new regional updates data
MaxItems	integer	1	Number of streaming regional updates stored in a cyclic buffer. Once this number of updates has been received, the oldest update is discarded whenever a new update arrives.
ClearBuffer	logical (true/false)	false	If true or 1, the internal cyclic quotes buffer is cleared after the data is returned to the caller.
AssignTo	string	" (empty string)	Contains the assignment target (typically a variable name) in the base Matlab workspace, useful for stream debugging.



Note: Regional updates data is only available in the Professional *IQC* license.

¹⁰⁷ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

6.3 Interval bars

The streaming interval bars feature has two parts, just like streaming ticks (§6.1):

1. Request IQFeed to start sending a stream of interval bars for a specified security. This is done by using the 'intervalbars' action and setting a **NumOfEvents** parameter to a positive >1 value.
2. At any later time(s), you can access the accumulated interval bars using either of the following alternatives:
 - a. Use 'intervalbars' action and **NumOfEvents** of -1 (minus one). This will return the latest streamed data, without stopping the background streaming.
 - b. If you set the **AssignTo** variable in the original request, you can directly access the streamed data in the specified base workspace variable.

For example, request 80 streaming interval bars of a continuous VIX future contract:

```
IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',80)
```

This causes IQFeed to start sending interval bars to *IQC* in the background, up to the specified **NumOfEvents**, without affecting normal Matlab processing. This means you can continue to work with Matlab, process data, display information etc.



Quotes will only stream in the background in non-blocking mode. If you assign the *IQC* command results to a variable, the request is treated as blocking and *IQC* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

```
IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',80); % streaming, non-blocking
data = IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',80); % blocking
```

NumOfEvents can be any number higher than 1 for streaming to work. To collect streaming quotes endlessly, set **NumOfEvents** to the value `inf`. Note that in Matlab, `inf` is a number (not a string), so do not enclose it in quotes ('inf').

The quotes are collected into an internal data buffer in *IQC*. A different buffer is maintained for each symbol. The buffer size can be controlled using the **MaxItems** parameter, which has a default value of 1. This means that by default only the latest streaming interval bar is stored.

If you set a higher value for **MaxItems**,¹⁰⁸ then up to the specified number of latest quotes will be stored, subject to IQFeed server limitations:¹⁰⁹

```
IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',80, 'MaxItems',3)
```



Note that using a large **MaxItems** increases memory usage. This may have an adverse effect if you use a large buffer (many thousands) and/or stream multiple **Symbols**.¹¹⁰

Each streaming security asset can have a different `BufferSize`, by specifying a different **MaxItems** value in the streaming command. This can be used for specifying a larger **MaxItems** for heavily-traded assets vs. lightly-traded ones.

¹⁰⁸ **MaxItems** is a numeric parameter like **NumOfEvents**, so don't enclose the parameter value within string quotes ('')

¹⁰⁹ The number of reported bars may possibly be limited by the IQFeed server, depending on your data subscriptions and exchange.

¹¹⁰ Interval bars use ~2KB of Matlab memory. So, if **MaxItems**=1000, each streamed symbol would need 1000*2KB = 2MB of Matlab memory when its buffer becomes full (which could take a while). Streaming 100 symbols will require up to ~200MB.

Subsequent requests to retrieve the latest accumulated interval bars buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one). These requests return a Matlab data struct similar to the following:

```
>> data = IQC('intervalbars', 'Symbol','@VX#', 'NumOfEvents',-1)
data =
    Symbol: '@VX#'
    Command: 'BW,@VX#,60,,1,3,,,B'
    isActive: 0
    EventsToProcess: 80
    EventsProcessed: 80
    LatestEventDatetime: 737902.504044143
    LatestEventTimestamp: '20200421 12:05:49.414'
    DataType: 'intervalbars'
    ProcessType: 'stream'
    AssignTo: ''
    errorMsg: ''
    MaxDaysToProcess: 1
    BufferSize: 3
    Buffer: [3x1 struct]
    LatestData: [1x1 struct]
```

In the returned data struct, we can see the following fields:

- **Symbol** – the requested **Symbol**.
- **Command** – the command sent to IQFeed, including the requested **Symbol**.
- **isActive** – logical flag indicating whether interval bars are currently streamed for the security. Once **NumOfEvents** bars are received this flag is set to false (0).
- **EventsToProcess** – total number of streaming interval bars requested for the security (using the **NumOfEvents** parameter).
- **EventsProcessed** – number of streaming interval bars received for this security. When **EventsProcessed** ≥ **EventsToProcess**, streaming is turned off and **isActive** is set to false (0). Note: it is possible that **EventsProcessed** > **EventsToProcess**, since it takes a while for the streaming cancellation request to reach IQFeed, and during this time a few additional bars may have arrived.
- **LatestEventDatetime** – Matlab numeric datetime representation of the **LatestEventTimestamp** (local timezone).
- **LatestEventTimestamp** – timestamp (string format) when this bar event was received by IQC (local timezone).
- **DataType** – always equal to 'intervalbars' for streaming interval bars.
- **ProcessType** – always equal to 'stream' for streaming interval bars.
- **AssignTo** – the name of the assigned variable in the base Matlab workspace (= **AssignTo** parameter, see below). In normal cases, this field is empty (").
- **errorMsg** – contains the error message in case streaming cannot be processed (for example: 'Symbol not found'). In normal cases, this field is empty (").
- **MaxDaysToProcess** – maximal number of days with intervals data to process.
- **BufferSize** – size of the data buffer (= **MaxItems** parameter, see below).
- **Buffer** – buffer of size **BufferSize**, accumulating the latest quote updates.
- **LatestData** – single latest interval bar received from IQFeed.

To retrieve the interval bars data, read the fields of the returned data struct:

```
>> data.LatestData
ans =
    Symbol: '@VX#'
    BarType: 'Complete interval bar from history'
    Timestamp: '2020-04-21 04:50:00'
        Open: 38.95
        High: 38.95
        Low: 38.85
        Close: 38.9
    CumulativeVolume: 6381
    IntervalVolume: 18
    NumberOfTrades: 0
        BarTypeCode: 'BH'
    EventDatetime: 737902.504044143
    EventTimestamp: '20200421 12:05:49.414'
```

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.¹¹¹

IQFeed sends interval bars asynchronously, with a `Timestamp` that identifies each bar (*server* timezone, typically US Eastern timezone). Note that `data.LatestEventDatetime=``data.LatestData.EventDatetime` and `data.LatestEventTimestamp=``data.LatestData.EventTimestamp` are specified in the *local* timezone (your computer's time). Also note that `LatestEventTimestamp` and `EventTimestamp` have a different format than `Timestamp`.

The `data.LatestData.NumberOfTrades` field indicates the number of trades that occurred within this bar (i.e., not cumulative). This field is only relevant when **IntervalType** is 'ticks'/trades'. In all other cases, the field will be empty ([]) or 0.

The **IntervalType** (default: 'secs') and **IntervalSize** (default: 60) parameters should typically be specified together. Note that **IntervalSize** must be a positive integer value (i.e. its value cannot be 4.5 or 0). If **IntervalType** is 'ticks'/trades', **IntervalSize** must be 2 or higher. If **IntervalType** is 'volume', **IntervalSize** must be 100 or higher. If **IntervalType** is 'secs', **IntervalSize** must be any integer between 1-300 (5 minutes), or any multiple of 60 (1 minute) between 300-3600 (1 hour), or 7200 (2 hours).¹¹²

The `data.LatestData.BarType` and `BarTypeCode` fields indicate if this is a historic bar (BH), a complete bar from the live (real-time) stream (BC), or an update bar (BU).



Unlike other streaming types, intervalbar queries also fetch historic bars data, starting from the date/time that is set by the **BeginDateTime** parameter. This is similar to (and subject to the same limitations as) historic interval data (§5.4), but with no specified end point. **BeginDateTime**'s default value is 00:00:00 today (server time), so we almost always get historic (BH) bars before live streaming (BC) bars. *IQC* will immediately return the historic bars, and new realtime streaming bars as they become available. If we run the query at mid-day, we may get hundreds of historic bars before the first live streaming bar. Depending on the specified **BeginDateTime**

¹¹¹ When **NumOfEvents** events have been received, IQFeed is instructed to stop streaming updates, but some update messages may already be on their way from IQFeed before streaming actually stops. These extra update messages are not accumulated in the `Buffer`, but the latest of these messages will be reflected in `LatestData` field.

¹¹² Note that IQFeed's limitations on live 'secs' interval bars are stricter than the limitations on historical interval bars (§5.4): <http://forums.dtn.com/index.cfm?page=topic&topicID=5529>

and **NumOfEvents**, we may receive only historic bars without any live streaming bars. Historic and streaming data bars can be distinguished based on their `BarType` field.

Note: the initial set of historic bars also includes the latest (incomplete) bar. For example, if we query at 21:34, we get the complete historic (BH) intervalbars for 21:20, 21:25, 21:30 and also the current (incomplete) bar for 21:35. A minute later, at exactly 21:35:00, IQFeed sends the complete BC bar for 21:35. This BC bar is added at the end of the buffer, so we will see two 21:35 bars in the buffer.

By default, IQFeed only sends historic BH bars followed by streaming BC bars. If the **MaxUpdateDuration**>0, IQFeed also sends BU bar update messages when the specified number of seconds have passed since the last trade or BC bar message. For example, if **IntervalSize**=60 and **MaxUpdateDuration**=15, the following scenario may occur:¹¹³

```
9:30:03 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:30:06 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:30:21 - No trade or BC message for 15 seconds, so IQFeed sends an update (BU) message
9:30:57 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:31:12 - No trade or BC message for 15 seconds, so IQFeed sends a new update (BU) message
9:31:15 - A trade occurs - this is the first trade outside the 9:30 minute, so IQFeed sends a bar complete (BC) message for the 9:30 bar. (15-sec timer is reset to 0)
9:31:29 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:31:40 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:31:53 - A trade occurs - no message is sent by IQFeed (15-sec timer is reset to 0)
9:32:03 - A trade occurs - IQFeed sends a bar complete (BC) message for the 9:31 bar; only 10 seconds have passed since the last trade, so no BU message is sent by IQFeed before the BC.
```

Note: IQFeed only sends BC bar complete messages when the first trade outside the bar occurs. This may occur a long time after the bar has ended. Use the **MaxUpdateDuration** parameter to receive more timely BU bar update messages.

BU update bars are appended to the buffer like other bar messages, so we may see bars with the same `Timestamp`. These bars do not necessarily have the same data, since they were sampled at different times (per their respective `EventDatetime`, `EventTimestamp`). The most updated data is always the last bar for each unique `Timestamp`. For example:

```
>> IQC('intervalbars', 'Symbol', '@VX#', 'NumOfEvents', inf, 'MaxItems', 9, ...
      'IntervalSize', 60, 'IntervalType', 'secs', 'MaxUpdateDuration', 15);

... % wait a while...
>> data = IQC('intervalbars', 'Symbol', '@VX#', 'NumOfEvents', -1);
>> struct2table(data.Buffer)
ans =
9x12 table
      Symbol      BarType      Timestamp      Open ...
      _____
'@VX#' 'Complete interval bar from history' '2020-04-21 03:51:00' 39.45 ...
'@VX#' 'Complete interval bar from history' '2020-04-21 03:52:00' 39.45 ...
'@VX#' 'Complete interval bar from history' '2020-04-21 03:53:00' 39.35 ...
'@VX#' 'Complete interval bar from stream' '2020-04-21 03:53:00' 39.35 ...
'@VX#' 'Updated interval bar' '2020-04-21 03:54:00' 39.45 ...
'@VX#' 'Complete interval bar from stream' '2020-04-21 03:54:00' 39.45 ...
'@VX#' 'Updated interval bar' '2020-04-21 03:55:00' 39.45 ...
'@VX#' 'Updated interval bar' '2020-04-21 03:55:00' 39.45 ...
'@VX#' 'Complete interval bar from stream' '2020-04-21 03:55:00' 39.45 ...
```

¹¹³ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=4341>. IQFeed's term "update interval" was renamed **MaxUpdateDuration** in *IQC* in order to avoid confusion with the interval bars. Note: the described scenario depends on IQFeed's implementation of the "update interval" mechanism – this implementation is not in *IQC*'s control.

Once the data is retrieved, you can direct *IQC* to clear (empty) the internal Buffer, by setting **ClearBuffer** to true or 1. The latest buffer will be returned, and the internal Buffer (but no other field) will be immediately emptied awaiting new interval bars:¹¹⁴

```
data = IQC('intervalbars', 'symbol', 'IBM', 'NumOfEvents', -1, ...
          'ClearBuffer', true);
```

To stop collecting interval bars, simply resend a request with **NumOfEvents=0**:

```
IQC('intervalbars', 'symbol', 'IBM', 'NumOfEvents', 0);
```

You can specify multiple symbols for streaming at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
IQC('intervalbars', 'symbols', {'IBM', 'GOOG', 'AAPL'}, 'numOfEvents', 6);
IQC('intervalbars', 'symbols', 'IBM:GOOG:AAPL', 'numOfEvents', 6); % equivalent
```

As with streaming quotes (§6.1), to get the latest data for all streamed symbols, omit the **Symbol** parameter or set it to empty []. This returns all streams (both active/not). In the following example, we get a struct array of size 5, one struct for each symbol for which that we have requested intervalbars during this Matlab session:

```
>> data = IQC('intervalbars', 'numOfEvents', -1); % no symbol: get ALL streams
data =
1x5 struct array with fields:
    Symbol
    Command
    isActive
    ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to ""):

```
>> IQC('intervalbars', 'numOfEvents', 0); % no symbol: stop ALL streams
```

When debugging streaming, it is sometimes useful to assign the streamed data to a base Matlab workspace variable, which can be monitored live using Matlab's Variable Explorer or directly checked/used within a Matlab script, without a need to call *IQC('intervalbars', ..., 'numOfEvents', -1)* to constantly refetch the latest data. This data mirroring can be done by setting the **AssignTo** parameter to any valid Matlab variable name or assignment target, for example: 'myData' or 'streamed.VX' or 'data{3}'.

```
>> IQC('intervalbars', ..., 'AssignTo', 'myData');
% streaming data in background is mirrored in base workspace variable myData:
>> myData
myData =
struct with fields:
    Symbol: '@VX#'
    Command: 'BW,@VX#,60,20200421 000000,7,100,,,B,s,,5'
    isActive: 1
    EventsToProcess: Inf
    EventsProcessed: 129
    LatestEventDatetime: 737902.831878183
    LatestEventTimestamp: '20200421 19:57:54'
    DataType: 'intervalbars'
    ProcessType: 'stream'
    AssignTo: 'myData'
    errorMsg: ''
    MaxDaysToProcess: 7
    BufferSize: 100
    Buffer: [100x1 struct]
    LatestData: [1x1 struct]
```

¹¹⁴ During and around the time of the buffer clear, some streaming data may be lost, so it is advised not to clear too often...

The following parameters affect interval bars data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ¹¹⁵	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0
NumOfEvents	integer	MaxItems	One of: <ul style="list-style-type: none"> • inf – continuous endless streaming interval bars for specified symbol(s) • N>1 – stream only N interval bars • 1 – get only a single interval bar • 0 – stop streaming interval bars • -1 – return latest interval bars data while continuing to stream new bars
MaxItems	integer	100	Returns up to the specified number of bars (if available). This is the max Buffer size.
MaxDays	integer	1	Max number of trading days to retrieve
IntervalType	string	'secs'	Sets the type of interval size. One of the following values: <ul style="list-style-type: none"> • 's' or 'secs' – time [seconds] (default) • 'v' or 'volume' – traded volume • 't' or 'ticks' – number of ticks
IntervalSize	integer	60	Size of bars in IntervalType units. Must be ≥ 1 for secs, ≥ 2 for ticks, ≥ 100 for volume.
BeginFilterTime	string	'00:00:00'	Only return bars that begin after this time of day (US Eastern time-zone). Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
EndFilterTime	string	'23:59:59'	Only return bars that end before this time of day (US Eastern time-zone). Format: 'hhmm', 'hh:mm', 'hhmmss' or 'hh:mm:ss'.
BeginDateTime	integer or string or datetime object	" (empty string) meaning today at 00:00:00	Only return bars that begin after this date/time (US Eastern time-zone). Format: Matlab datenum, or 'yyyymmdd hhmmss', or 'yyyy-mm-dd hh:mm:ss' etc. Note: there is no corresponding EndDateTime parameter for streaming intervalbars (only for historic bars: §5.4).

¹¹⁵ In IQC, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
MaxUpdateDuration	integer	0	Max number of seconds to wait after a trade before receiving a bar update message
Timeout	number	5.0	Max number of seconds to wait (0-9000) for data in blocking mode (0 means infinite)
ClearBuffer	logical (true/false)	false	If true or 1, the internal cyclic quotes buffer is cleared after data is returned to the caller
AssignTo	string	" (empty string)	Contains the assignment target (typically a variable name) in the base Matlab workspace, useful for stream debugging.

6.4 Market depth (Level 2)

The streaming market depth mechanism also has two distinct parts, just like streaming level 1 quotes (§6.1):

1. Request IQFeed to start sending a stream of market depth quotes for a specified security. This is done by using the 'marketdepth' action.
2. At any later time(s), you can access the current market depth data using either of the following alternatives:
 - a. Use 'marketdepth' action and **NumOfEvents** of -1 (minus one). This will return the latest streamed data, without stopping the background streaming.
 - b. If you set the **AssignTo** variable in the original request, you can directly access the streamed data in the specified base workspace variable.

For example, let's request market depth quotes for a continuous E-mini contract:

```
IQC('marketdepth', 'Symbol', '@ES#')
```

This causes IQFeed to start sending market depth updates to *IQC* in the background, up to the specified **NumOfEvents**, without affecting normal Matlab processing. This means you can continue to work with Matlab, process data, display information etc.

Note that each incoming quote message updates the data for a single market depth row. The market depth row cannot be specified nor predicted by the user, and the order of messages is unrelated to the market depth row, for example, an update for row #3 can follow an update of row #5.



Market depth data will only stream in the background in non-blocking mode. If you assign the *IQC* command results to a variable, the request is treated as blocking and *IQC* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

```
IQC('marketdepth', 'Symbol', '@ES#', 'NumOfEvents', 600); % streaming, non-blocking
data = IQC('marketdepth', 'Symbol', '@ES#', 'NumOfEvents', 600); % blocking
```

NumOfEvents is an optional input parameter and can be any number >1 for streaming. To collect market depth data endlessly, set **NumOfEvents** to the value `inf`. Note that in Matlab, `inf` is a number (not a string), so do not enclose it in quotes ('inf').

The quotes are collected into an internal data structure in *IQC*. A different structure is maintained for each symbol.

Subsequent requests to retrieve the latest accumulated interval bars buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one). These requests return a Matlab data struct similar to the following:

```
>> data = IQC('marketdepth', 'Symbol', '@ES#', 'NumOfEvents', -1)
data =
    Symbol: '@ES#'
    Command: 'w@ES#'
    EventsToProcess: 600
    EventsProcessed: 437
    LatestEventDatenum: 737195.518211377
    LatestEventTimestamp: '20180515 12:26:13'
    AssignTo: ''
    errorMsg: ''
    IncludeEmptyQuotes: 0
    LatestData: [1x10 struct]
```

In the returned `data` struct, we can see the following fields:

- `Symbol` – the requested Symbol.
- `Command` – the command sent to IQFeed, including the requested Symbol.
- `EventsToProcess` – total number of streaming interval bars requested for the security (using the **NumOfEvents** parameter).
- `EventsProcessed` – number of streaming market depth data quotes received for this security. When `EventsProcessed >= EventsToProcess`, streaming market depth data for this security is turned off.
- `LatestEventDatetime` – Matlab numeric datetime representation of the `LatestEventTimestamp` (local timezone).
- `LatestEventTimestamp` – timestamp (string format) when this market depth quote event was received by IQC (local timezone).
- `AssignTo` – the name of the assigned variable in the base Matlab workspace (=AssignTo parameter, see below). In normal cases, this field is empty (").
- `errorMsg` – contains the error message in case streaming cannot be processed (for example: 'Symbol not found'). In normal cases, this field is empty (").
- `IncludeEmptyQuotes` – value of the specified **IncludeEmptyQuotes** parameter (default value: false). If true or 1, then empty quotes (having no valid Bid or Ask) will be reported and be counted as a valid “event”; otherwise they will not.
- `LatestData` – latest data received by IQFeed for each market depth row.

To retrieve the market depth data at the n^{th} market depth row, simply read the fields of the `LatestData` at the n^{th} location, for example:

```
>> data.LatestData(4)
ans =
        Symbol: '@ES#'
           ID: 'MD04'
         Bid: 2725.5
         Ask: 2727.25
        BidSize: 65
        AskSize: 148
        BidTime: '05:25:59.761191'
           Date: '2018-05-15'
        AskTime: '05:25:59.760278'
    BidInfoValid: 1
    AskInfoValid: 1
        Condition: 52
    Condition_Description: 'regular'
        ID_Description: 'Order book row #4'
```

`BidInfoValid` and `AskInfoValid` values are logical (true/false) values, which appear as 1 or 0, respectively, in the struct display above. The `ID` field indicates the corresponding order-book row (for futures) or market-maker name (for equities).

Different market depth quotes are sent independently from the IQFeed server with a unique timestamp, in a non-ordered manner. Note that `data.LatestEventDatetime` and `data.LatestEventTimestamp` are specified in the local time-zone.



Note: unlike streaming quotes (§6.1), regional updates (§6.2), and interval bars (§6.3), the streaming market depth mechanism does not store an internal buffer of quote updates, so there is no `Buffer` field and previous updates are not stored. Only the latest snapshot of the order book (in the `LatestData` field) is updated.

To stop collecting market depth quotes for a security, simply send the request again, this time with **NumOfEvents=0**.

```
IQC('marketdepth', 'Symbol','@ES#', 'NumOfEvents',0);
```

You can specify multiple symbols for streaming at the same time, in a single *IQC* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
IQC('marketdepth', 'symbols',{'IBM','GOOG','AAPL'});
IQC('marketdepth', 'symbols','IBM:GOOG:AAPL'); % equivalent
```

As with the blocking request (§4.4), you'll receive an error message when requesting market depth info from an exchange for which you have no Level 2 data subscription:

```
>> data = IQC('marketdepth', 'Symbol','IBM', ...) %not subscribed to NYSE L2
Error using IQC
Symbol 'IBM' was not found!
```

As with streaming quotes (§6.1), to get the latest data for all streamed symbols, omit the **Symbol** parameter or set it to empty []. This returns all streams (both active/not):

```
>> data = IQC('marketdepth', 'numOfEvents',-1); % no symbol: get ALL streams
data =
1x5 struct array with fields:
    Symbol
    Command
    isActive
    EventsToProcess
    ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to ""):

```
>> IQC('marketdepth', 'numOfEvents',0); % no symbol: ALL streams are stopped
```

When debugging streaming, it is sometimes useful to assign the streamed data to a base Matlab workspace variable, which can be monitored live using Matlab's Variable Explorer or directly checked/used within a Matlab script, without a need to call `IQC('marketdepth',...,'numOfEvents',-1)` to constantly refetch the latest data. This data mirroring can be done by setting the **AssignTo** parameter to any valid Matlab variable name or assignment target, for example: 'myData' or 'streamed.VX' or 'data{3}'.

```
>> IQC('marketdepth', ..., 'AssignTo','myData');
% streaming data in background is mirrored in base workspace variable myData:
>> myData
myData =
struct with fields:
    Symbol: '@ES#'
    Command: 'w@ES#'
    EventsToProcess: 600
    EventsProcessed: 437
    LatestEventDatetime: 737195.518211377
    LatestEventTimestamp: '20180515 12:26:13'
    AssignTo: 'myData'
    errorMsg: ''
    IncludeEmptyQuotes: 0
    LatestData: [1x10 struct]
```

The following parameters affect market depth data queries:

Parameter	Data type	Default	Description
Symbol or Symbols ¹¹⁶	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: <ul style="list-style-type: none"> • '@ES#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when NumOfEvents >0
NumOfEvents	integer	10 ¹¹⁷	One of: <ul style="list-style-type: none"> • inf – continuous endless streaming Level 2 data for specified symbol(s) • N>1 – only process N incoming quotes • 1 – get only a single quote • 0 – stop streaming market depth data • -1 – return the latest Level 2 data while continuing to stream new data updates
IncludeEmpty Quotes	logical (true/false)	false	If set to true or 1, empty Level 2 quotes (with neither a valid Bid nor valid Ask) will also be returned. By default (false), they will not be. See §4.4 for details.
ClearBuffer	logical (true/false)	false	If true or 1, the internal cyclic quotes buffer is cleared after the data is returned to the caller.
AssignTo	string	" (empty string)	Contains the assignment target (typically a variable name) in the base Matlab workspace, useful for stream debugging.



Note: Market Depth (Level 2) data is only available in the Professional *IQC* license.

¹¹⁶ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

¹¹⁷ The default value of **NumOfEvents** was changed from `inf` to 10 in *IQC* version 2.17 (released on 2019-05-05)

7 News

News headlines and stories can be retrieved via the 'news' action. Several data-types are available, which can be set using the **DataType** parameter.



Note: News data is only available in the Professional *IQC* license.

7.1 Configuration

To retrieve the news configuration for your account, set **DataType** to 'config':

```
>> data = IQC('news', 'DataType', 'config')
data =
  Category: 'All News'
  Majors: [1x7 struct]
>> {data.Majors.Source}
ans =
  1x7 cell array
    {'DTN'}    {'CPR'}    {'CBW'}    {'RTT'}    {'CPZ'}    {'CIW'}    {'BEN'}
>> {data.Majors.Description}
ans =
  1x7 cell array
    {'DTN News'}    {'PR Newswire'}    {'Business Wire'}    {'Real-Time Trader'}
    {'GlobeNewswire Inc'}    {'Marketwire'}    {'Benzinga Pro'}
```

This shows that we are connected to 7 major news sources. We can drill-down for details about these news sources:

```
>> data.Majors(1)
ans =
    Source: 'DTN'
  Description: 'DTN News'
  AuthenticationCode: '1D'
    IconID: 10
   Minors: [1x4 struct]
>> data.Majors(1).Minors(1)
ans =
    Source: 'DT5'
  Description: 'Treasuries, Most Actives, Gainers, Losers'
  AuthenticationCode: '1D'
>> data.Majors(1).Minors(2)
ans =
    Source: 'RTL'
  Description: 'Derivatives - Selected Futures and Options'
  AuthenticationCode: '2Ab'
    IconID: 10
```

Note that some news sources have no “Minor” news-sources:

```
>> data.Majors(2)
ans =
    Source: 'CPR'
  Description: 'PR Newswire'
  AuthenticationCode: '1X'
    IconID: 5
   Minors: [1x0 struct]
>> data.Majors(7)
ans =
    Source: 'BEN'
  Description: 'Benzinga Pro'
  AuthenticationCode: '1a'
    IconID: 10
   Minors: [1x0 struct]
```

News configuration queries do not have any user-settable parameters.

7.2 Story headlines

To retrieve the latest news headlines (in blocking mode), set **DataType** to 'headlines':

```
>> data = IQC('news', 'DataType', 'headlines')
data =
    1000x1 struct array with fields:
        Source
        ID
        Symbols
        Timestamp
        Text
        Story
>> data(1)
ans =
    Source: 'CPR'
    ID: 21988707473
    Symbols: {}
    Timestamp: 20180305064553
    Text: 'The Surface Disinfectants Market is Expected to Grow at a CAGR
    of 8.3% to a USD '
    Story: ''
>> data(2)
ans =
    Source: 'BEN'
    ID: 21988707468
    Symbols: {'BZFDA' 'CVRS'}
    Timestamp: 20180305064533
    Text: 'Corindus Receives FDA Clearance for First Automated Robotic
    Movemen...'
    Story: ''
>> data(3)
ans =
    Source: 'RTB'
    ID: 21988701358
    Symbols: {'BSX'}
    Timestamp: 20180305064233
    Text: 'Boston Scientific Corp Q4 adjusted earnings Miss Estimates'
    Story: ''
```

As can be seen, some stories are specific to particular symbols (BZFDA and CVRS in story #21988707468, BSX in #21988701358), while others are not (#21988707473).

Also note that the news stories' `Timestamp` is reported in `yyyymmddHHMMSS` format, where the time is specified in US Eastern time-zone.

When you retrieve news headlines, you might run into a timeout problem: by default, IQFeed send the latest 1000 news headlines and only some of them might be received by *IQC* before the built-in **Timeout** (default: 5 secs) forces *IQC* to return the data to the user (remember, this is blocking mode, where a timeout applies):

```
>> data = IQC('news', 'DataType', 'headlines')
Warning: IQC timeout: only partial data is returned. Perhaps the Timeout
parameter should be set to a value larger than 5 or the NumOfEvents parameter
to a value smaller than Inf
data =
    738x1 struct array with fields:
        Source
        ID
        Symbols
        Timestamp
        Text
```

As suggested by the message, you can set the **Timeout** parameter to a high value in order to allow *IQC* more time to gather the data before returning the results: ¹¹⁸

```
>> data = IQC('news', 'DataType','headlines', 'Timeout',10)
data =
    1000x1 struct array with fields:
        Source
        ID
        ...
```

You can filter the headlines to a specific set of symbols by specifying **Symbols** as a colon-delimited or cell-array list of symbols. ¹¹⁹ For example, to filter only headlines that relate to symbols BSX, BSX/AAPL, and BSX/AAPL/GOOG, respectively:

```
>> data = IQC('news', 'DataType','headlines', 'Symbols','BSX')
data =
    60x1 struct array with fields:
        ...
>> data = IQC('news', 'DataType','headlines', 'Symbols',{'BSX','AAPL'})
data =
    677x1 struct array with fields:
        ...
>> data = IQC('news', 'DataType','headlines', 'Symbols','BSX:AAPL:GOOG')
data =
    841x1 struct array with fields:
        ...
```

You can also limit the search to specific news sources, by specifying a colon-separated or cell-array list of sources in the **Sources** parameter. For example:

```
>> data = IQC('news', 'DataType','headlines', 'Symbols','BSX:GOOG:AAPL', ...
              'Sources','DTN:CPR:BEN')
data =
    745x1 struct array with fields:
        ...
```

In this example, we see that when we limit our search to DTN (DTN News), CPR (PR Newswire), and BEN (Benzinga Pro), we only get 745 headlines, compared to 841 headlines from all the news sources. The news source names are the ones reported by the `Majors.Source` field, in the news configuration query (see §7.1).

In addition to limiting the search to a certain news source, you can also limit it to certain meta-tags that are assigned by some news sources, using the **Symbols** parameter. For example, to limit the search to “Benzinga Ratings”:

```
>> data = IQC('news', 'DataType','headlines', 'Symbols','BZRatings');
```

You can limit the reported headlines to only a specific date, using the **Date** parameter:

```
>> data = IQC('news', 'DataType','headlines', 'Date',20180304, ...
              'Symbols',{'BSX','AAPL'})
data =
    14x1 struct array with fields:
        ID
        ...
```

Date can be specified in various formats: as a Matlab `datetime` object, a numeric Matlab datenum (737089), a numeric `yyyymmdd` value (20180129), or a string ('2018/01/29', '2018-01-29' or '20180129'). Note: IQFeed only stores headlines of the past 180 days. ¹²⁰

¹¹⁸ The **Timeout** parameter is automatically set to a minimal value of 60 [secs] when **GetStory** parameter is requested (see below)

¹¹⁹ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

You can also limit the maximal number of reported headlines using the **MaxItems** parameter. This will report the latest **MaxItems** news headlines (fewer headlines may actually be reported, depending on their availability):¹²⁰

```
>> data = IQC('news', 'DataType','headlines', 'MaxItems',50)
data =
    50x1 struct array with fields:
        Source
        ID
        ...
```

By default, only the headline text is returned. To also fetch the full news-story text that is associated with each headline, set **GetStory** to `true`:

```
>> data = IQC('news', 'DataType','headlines', 'GetStory',true);
>> data(1)
ans =
    Source: 'CBW'
    ID: 22017456356
    Symbols: {}
    Timestamp: '20180524 092926'
    Text: 'Global Barium Nitrate Market - Emergence of Environment-
    Friendly Ox...'
    Story: '09:28 Thursday, May 24, 2018. (RTTNews.com) - Babcock & Wilcox
    Enterprises, Inc. (BW) confirmed that it had received a non-binding indication
    of interest from Steel Partners to acquire B&W in a transaction in which B&W...
    For comments and feedback: contact editorial@rttnews.com Copyright(c) 2018
    RTTNews.com All Rights Reserved'
```

Each news story takes 0.3-0.5 secs to download, so querying the story text for multiple headlines can take a long time. For example, such a query might take a full minute for 100 headlines. If you have Matlab's Parallel Computing Toolbox and the Professional *IQC* license, you can parallelize this query by setting **UseParallel** to `true`:

```
>> tic
>> data = IQC('news', 'DataType','headlines', 'MaxItems',100, 'GetStory',1);
>> toc
Elapsed time is 56.311768 seconds.
>> parpool('local',4) % start 4 workers in parallel pool (optional)
>> tic
>> data = IQC('news', 'DataType','headlines', 'MaxItems',100, 'GetStory',1,...
    'UseParallel',1);
>> toc
Elapsed time is 15.799185 seconds.
```

News headlines queries with **GetStory** are composed of an initial headlines query, followed by multiple news-story queries (§7.3) for each of the reported headline IDs. Such queries with **GetStory** have an automatic minimal **Timeout** value of 60 [secs]. The **Timeout** parameter has a practical effect only on the initial headlines query, not the subsequent story text queries, which take much longer in total. A **Timeout** of 60 (the default/min value for such queries) allows fetching up to ~10-20k headlines in theory (depending on network/computer speed and **MaxItems/Date** parameter values), but only up to 4k headlines are returned in practice (IQFeed's undocumented limit). The news-story queries for these 4k headlines could take an extra 20-40 minutes (depending on network/computer speed), unless you parallelize the query. This would *not* be cut short by **Timeout**, since **Timeout** affects each news-story query separately. You can only stop the query by typing Ctrl-C in Matlab's console (Command Window).

¹²⁰ <https://help.dtniq.com/support-faqs>

¹²¹ IQFeed ignores **MaxItems**>4000, returning only 4000 headlines: <http://forums.dtn.com/index.cfm?page=topic&topicID=5702>

The following parameters affect (filter) news headlines queries:

Parameter	Data type	Default	Description
Symbol or Symbols ¹²²	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits the query to the specified symbols and meta-tags only (or to all symbols, if empty). Examples: <ul style="list-style-type: none"> 'IBM' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM', 'AAPL', 'GOOG'} 'BZRatings:BZTradingIdeas'
Sources	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits query to the specified news sources only (or to all sources, if empty). Examples: <ul style="list-style-type: none"> 'DTN' 'DTN:CPR:BEN' 'DTN,CPR,BEN' {'DTN', 'CPR', 'BEN'}
Date	integer or string or datetime object	[] meaning all	Date at which the news headline was published (or all dates, if empty). Examples: <ul style="list-style-type: none"> 737089 (Matlab datenum format) datetime('Jan 29, 2018') 20180129 (yyyymmdd format) '20180129' '2018/01/29' '2018-01-29'
MaxItems	integer (1-4000) ¹²³	1000	Maximal # of headlines to be reported by IQFeed. Note that a lower number of headlines may be reported, depending on their availability, based on the other filters.
GetStory	logical (true/false)	false	If false (default), only store the incoming headline messages. If true or 1, automatically fetch and store the full story text for each incoming headline. Parallelizable using UseParallel (see below).
Timeout	number	5.0	Max # of seconds to wait for incoming data (0-9000 where 0 means infinite). If GetStory was requested Timeout is set to minimum 60.
UseParallel	logical (true/false)	false	If set to true or 1, and if Parallel Computing Toolbox is installed, then querying story headlines using GetStory=true is done in parallel (see §3.6; Pro IQC license only).
MaxWorkers	integer	(the current pool size)	Max number of parallel workers to use (up to the current pool size) when UseParallel=1

¹²² In IQC, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

¹²³ IQFeed ignores **MaxItems**>4000, returning only 4000 headlines: <http://forums.dtn.com/index.cfm?page=topic&topicID=5702>

7.3 Story text

To read a particular story in full (blocking mode), specify **DataType** = 'story' and **ID** (numeric ID, as provided in the story-headlines query, see §7.2). Different news sources provide their news stories in different formats, for example:

```
>> data = IQC('news', 'DataType','story', 'ID',21988707468)
data =
    ID: 21988707468
    Symbols: {'BZFDA' 'CVRS'}
    Text: 'Corindus Receives FDA Clearance for First Automated Robotic
          Movement in technIQ Series for CorPath GRX Platform.'
```

```
>> data = IQC('news', 'DataType','story', 'ID',21988701358)
data =
    ID: 21988701358
    Symbols: {'BSX'}
    Text: '06:42 Monday, March 05, 2018. (RTTNews.com) - Boston Scientific
          Corp (BSX) released earnings for fourth quarter that declined
          from the same period last year... % full text redacted here
          Read the original article on RTTNews
          (http://alpha.rttnews.com/9583/boston-scientific-corp-q4-
          adjusted-earnings-miss-estimates.aspx) For comments and
          feedback: contact editorial@rttnews.com. Copyright(c) 2018
          RTTNews.com All Rights Reserved.'
```

In many cases, the news story is not specifically related to any particular symbol:

```
>> data = IQC('news', 'DataType','story', 'ID',21991159700)
data =
    ID: 21991159700
    Symbols: {}
    Text: 'Global Nanocatalysts Strategic Business Report 2018: Market
          Trends, Growth Drivers & Issues 2016-2024 -
          ResearchAndMarkets.com. Mar. 12, 2018. Business Editors. DUBLIN-
          -(BUSINESS WIRE)--Mar. 12, 2018--The Nanocatalysts - Global
          Strategic Business Report... % full text redacted here
          View source version on businesswire.com:
          http://www.businesswire.com/news/home/20180312005490/en/ ...
          For GMT Office Hours Call +353-1-416-8900. Related Topics:
          Nanotechnology, Nanomaterials'
```

In some cases, the story may be assigned one or more meta-symbol tags. For example, the following story is tagged for “Benzinga Ratings”:

```
>> data = IQC('news', 'DataType','story', 'ID',21991162633)
data =
    ID: 21991162633
    Symbols: {'BZRatings' 'MNTX'}
    Text: 'Manitex International Sees Q4 Sales $64.40M vs $64.45M Est.
          Manitex International (NASDAQ: MNTX) sees Q4 sales of $64.40M
          vs $64.45M estimate.'
```

Note that separate paragraphs in the news story text are separated by a newline (char(10)) in the reported `data.Text` field. This enables display of the story text in a human-readable format, when you output the text to the Matlab console or GUI.

If the requested **ID** is invalid or does not exist, the returned data will be empty (no error is reported):

```
>> IQC('news', 'DataType', 'story', 'ID', 123456) % non-existing headline ID
ans =
    []
```

Aside from **ID**, the news story-text query does not have any user-settable parameters.

You can specify multiple **IDs** in a single *IQC* query command, by specifying an array of values. For example:

```
>> data = IQC('news', 'DataType', 'story', 'ID', [22018991229, 22018991585])
data =
    2x1 struct array with fields:
        ID
        Symbols
        Text

>> data(1)
ans =
        ID: 22018991229
    Symbols: {}
        Text: 'May 29, 2018 ↵Dublin, May 29, 2018 (GLOBE NEWSWIRE) -- The
European Financing in Cleantech Innovation report...'

>> data(2)
ans =
        ID: 22018991585
    Symbols: {'BZEarnings' 'MOMO'}
        Text: 'Momo Inc. Earlier Reported Q1 EPS $0.69 Beat $0.50 Estimate,
Sales $435.129M Beat $396.17M Estimate ↵Momo Inc. ...'
```

7.4 Story count

It is sometimes useful to know the number of distinct news stories, from all news sources (even those to which you are not subscribed), that relate to different symbols, indicating level of news interest in those symbols. Set **DataType** to 'number' and the **Symbols**, **Sources** and/or dates, to receive a Matlab struct with a numeric count for each symbol:

```
>> data = IQC('news', 'DataType','number', 'Symbols','BSX')
data =
    BSX: 14
>> data = IQC('news', 'DataType','number', 'Symbols','BSX:HP:AAPL:GOOG')
data =
    AAPL: 7
    BSX: 14
    GOOG: 2
    HP: 0
```

In this example, we see that BSX has a higher news-count today than AAPL or GOOG. Symbols having no news items will appear at the bottom of the struct with a count of 0.

You can limit the search to specific news sources, by specifying a colon-separated or cell-array list of sources in the **Sources** parameter. For example:

```
>> data = IQC('news', 'DataType','number', 'Symbols','BSX:GOOG:AAPL',...
               'Sources','DTN:CPR:BEN')
data =
    AAPL: 2
    BSX: 3
```

In this example, we see that when we limit our search to DTN (DTN News), CPR (PR Newswire), and BEN (Benzinga Pro), AAPL and BSX have fewer news items, and GOOG has none. The news source names are the ones reported by the `Majors.Source` field, in the news configuration query (see §7.1).

You can also filter the search to only look at news items published at specific dates, by specifying the **BeginDate**, **EndDate** and/or **Date** parameters. Dates can be specified in several formats: as a Matlab `datetime` object, Matlab numeric datenum (737089), numeric `yyyymmdd` (20180129), or string ('2018/01/29', '2018-01-29', '20180129'):

```
>> data = IQC('news', 'DataType', 'number', 'Symbols','BSX:GOOG:AAPL',...
               'BeginDate',20180301)
data =
    AAPL: 45
    BSX: 19
    GOOG: 15
>> data = IQC('news', 'DataType', 'number', 'Symbols','BSX:GOOG:AAPL',...
               'BeginDate',20180301, 'EndDate',20180303)
data =
    AAPL: 37
    BSX: 3
    GOOG: 13
>> data = IQC('news', 'DataType', 'number', 'Symbols','BSX:GOOG:AAPL',...
               'EndDate',20180305)
data =
    AAPL: 2038
    BSX: 191
    GOOG: 996
>> data = IQC('news', 'DataType', 'number', 'Symbols','BSX:GOOG:AAPL',...
               'Date',20180301)
data =
    AAPL: 16
    BSX: 1
    GOOG: 3
```

IQC returns a Matlab struct, so the reported symbols need to be valid field names, and non-alphanumeric characters are automatically converted. For example:

```
>> data = IQC('news', 'DataType','number', 'Symbols','BOL.ST:BOL@SS:0QLL.L')
data =
    x0QLL_L: 3
    BOL_ST: 1
    BOLxSS: 1
```

The following parameters affect (filter) news story-count queries:

Parameter	Data type	Default	Description
Symbol or Symbols ¹²⁴	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits query to specified symbols, meta-tags only (or to all symbols, if empty). Examples: <ul style="list-style-type: none"> 'IBM' 'IBM:AAPL:GOOG' {'IBM', 'AAPL', 'GOOG'} 'BZRatings,BZTradingIdeas'
Sources	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits the query to specified news sources only (or to all sources, if empty). Examples: <ul style="list-style-type: none"> 'DTN' 'DTN:CPR:BEN' 'DTN,CPR,BEN' {'DTN', 'CPR', 'BEN'}
Date	integer or string or datetime object	[] meaning today	Specific date at which the news items were published. Examples: <ul style="list-style-type: none"> 737089 (Matlab datenum format) datetime('Jan 29, 2018') 20180129 (yyyymmdd format) '20180129' '2018/01/29' '2018-01-29' Note: Date overrides BeginDate , EndDate
BeginDate	integer or string or datetime object	'1900/01/01' (i.e., from as early as data is available)	Earliest date at which the news items were published. Examples: see Date above.
EndDate	integer or string or datetime object	'2099/12/31' (i.e., until now)	Latest date at which the news items were published. Examples: see Date above.

¹²⁴ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

7.5 Streaming news headlines

The streaming news mechanism has two parts, just like streaming ticks (§6.1):

1. Request IQFeed to start sending a stream of news headlines. This is done by using the 'news' action and setting a **NumOfEvents** parameter to a positive >1 value. You can limit the headlines to certain news source(s) using the **Sources** parameter, and/or to certain symbol(s) using the **Symbols** parameter.
2. Later, whenever you wish to process the latest news headline(s), simply use the 'news' action and **NumOfEvents** of -1 (minus one). This will return the latest information (a data struct), without stopping the background streaming.

For example, let's request 100 streaming headlines for Facebook and Apple:

```
IQC('news', 'Symbols', 'FB:AAPL', 'NumOfEvents', 100)
```

This causes IQFeed to start sending news headlines to *IQC* in the background, up to the specified **NumOfEvents**, without affecting normal Matlab processing. This means that you can continue to work with Matlab, process and display information etc.



Headlines will only stream in the background in non-blocking mode. If you assign the *IQC* command results to a variable, the request is treated as blocking and *IQC* will wait for all the events to accumulate (or **Timeout** to occur), as described in §7.2:

```
IQC('news', 'NumOfEvents', 100);           % streaming, non-blocking
data = IQC('news', 'NumOfEvents', 100);    % blocking
```

NumOfEvents can be any number higher than 1 for streaming to work (a value of 1 is the standard snapshot news-headline request described in §7.2). To collect streaming headlines endlessly, set **NumOfEvents** to the value `inf`. Note that in Matlab, `inf` is a number (not a string), so do not enclose it in quotes ('`inf`').

The headlines are collected into an internal data buffer in *IQC*. Unlike streaming quotes, all headlines, for all symbols, are collected in a single buffer. The buffer size can be controlled using the **MaxItems** parameter, which has a default value of `inf`¹²⁵. This means that by default all the streaming headlines that affect the specified symbols will be stored in the buffer and become accessible for later processing.¹²⁶

If you set a higher value for **MaxItems**, then up to the specified number of latest news headline items will be stored. For example, to store the latest 50 headlines:

```
IQC('news', 'NumOfEvents', 100, 'MaxItems', 50)
```



Note that using a large **MaxItems** increases memory usage, which could have an adverse effect if you set a very large buffer size (many thousands) and/or streaming for a large number of different securities.¹²⁷

¹²⁵ Note that this too is different from the streaming quotes mechanism, where the default **MaxItems** value is 1. Note that **MaxItems** is a numeric parameter like **NumOfEvents**, so don't enclose the parameter value within string quotes ('')

¹²⁶ This might have a memory and performance implication if you leave streaming news on for a long time, for a large number of symbols. See the discussion of memory and performance implications further below.

¹²⁷ Each news headline item uses 1-2KB of Matlab memory. During trading hours, there could be 10-20 headlines per minute for all symbols (i.e., 1K headlines, or 1-2MB per hour, unless you limit **Symbols** to certain symbols). Limiting **Symbols** to certain symbols and/or setting **MaxItems** to some finite value, ensures that memory usage and performance impact remain low.

Subsequent requests to retrieve the latest accumulated headlines buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one). These requests return a Matlab data struct similar to the following:

```
>> data = IQC('news', 'NumOfEvents', -1)
data =

    Command: 'S,NEWSON'
    isActive: 1
    EventsToProcess: 100
    EventsProcessed: 13
    LatestEventDatetime: 737146.726041343
    LatestEventTimestamp: '20180327 17:25:29'
    DataType: 'news'
    ProcessType: 'stream'
    Sources: {}
    Symbols: {}
    BufferSize: 50
    Buffer: [13x1 struct]
    LatestData: [1x1 struct]
```

In the returned data struct, we can see the following fields:

- **Command** – the command sent to IQFeed.¹²⁸
- **isActive** – a flag indicating whether headlines are currently being streamed. When **NumOfEvents** ticks have been received, this flag is set to *false* (0).
- **EventsToProcess** – total number of streaming headlines requested (using the **NumOfEvents** parameter).
- **EventsProcessed** – number of streaming headlines received. When **EventsProcessed** ≥ **EventsToProcess**, streaming headlines are turned off and **isActive** is set to *false* (0). Note that it is possible that **EventsProcessed** > **EventsToProcess**, since it takes a while for the streaming cancellation request to reach IQFeed and during this time a few additional items may have arrived.
- **LatestEventDatetime** – Matlab numeric datetime representation of the **LatestEventTimestamp**.
- **LatestEventTimestamp** – local timestamp (string format) when this headline was received by IQC.
- **DataType** – always equal to 'news' for streaming headlines.
- **ProcessType** – always equal to 'stream' for streaming headlines.
- **Sources** – cell array of acceptable news sources, set by the **Sources** parameter. Headline events from all other sources are ignored. When **Sources** is empty, no headline is ignored based on its source.
- **Symbols** – cell array of acceptable symbols, set by the **Symbols** parameter. Headline events that affect all other symbols are ignored. When **Symbols** is empty, no headline is ignored based on its related symbol(s).
- **BufferSize** – size of the data buffer (= **MaxItems** parameter, see below).
- **Buffer** – buffer of size **BufferSize**, accumulating the latest headline updates.
- **LatestData** – latest headline event received from IQFeed.

¹²⁸ Note that this is not specific to symbols/sources: filtering based on symbol/source is done on the incoming headline messages.

To get the headline data, read the fields of the returned `data` struct, for example:

```
>> data.LatestData
ans =
    Source: 'BEN'
      ID: 21996096022
  Symbols: {'BZRatings' 'FB'}
Timestamp: '20180326 083326'
      Text: 'Baird Maintains Outperform on Facebook Lowers Price Target to $210'
      Story: ''
```

Each headline has an associated timestamp, since different headlines are sent separately and independently from IQFeed server.

By default, **GetStory** is set to `false`, resulting in empty `data.LatestData.Story`. To automatically retrieve the full story text associated with each streamed headline, set **GetStory** to `true` (see §7.2). In any case, it is always possible to retrieve individual story texts using their headline ID (see §7.3).

Note: while `data.LatestEventDatetime` and `data.LatestEventTimestamp` are specified in the local time-zone, `data.LatestData.Timestamp` is specified in the server's time-zone.

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.¹²⁹

To stop collecting streaming headlines for a security, simply send the request again, this time with **NumOfEvents**=0.

You can specify one or more symbols for streaming, by specifying a colon-delimited or cell-array list of symbols. If **Symbols** is specified, then any headline that does not relate to one or more of the specified **Symbols** will be ignored (skipped). For example:

```
IQC('news', 'symbols', {'IBM', 'GOOG', 'AAPL'}, 'numOfEvents', 6);
IQC('news', 'symbols', 'IBM:GOOG:AAPL', 'numOfEvents', 6); % equivalent
```

You can also specify meta-tags assigned by some news sources. For example, to limit streaming headlines to “Benzinga Ratings” and anything related to Facebook or Apple:

```
IQC('news', 'Symbols', 'BZRatings:FB:AAPL', 'numOfEvents', 6);
```

Note: if you omit the **Symbols** parameter in your *IQC* command, no filtering of headlines based on affected symbols is performed, and all headlines will be collected.

Similarly, you can specify one or more news sources, by specifying a colon-delimited or cell-array list of sources. If **Sources** is specified, then any headline that does not originate from one of the specified **Sources** will be ignored and will not be recorded:

```
IQC('news', 'sources', {'DTN', 'CPR', 'BEN'}, 'numOfEvents', 6);
IQC('news', 'sources', 'DTN:CPR:BEN', 'numOfEvents', 6); % equivalent
```

¹²⁹ When **NumOfEvents** events have been received, IQFeed is instructed to stop streaming updates, but one or more update messages may already be on their way from IQFeed before streaming actually stops. These extra update messages are not accumulated in the `Buffer`, but the latest of these messages will be reflected in `LatestData` field.

As before, if you omit the **Sources** parameter in your *IQC* command, no filtering of headlines based on their source will be performed, and all headlines will be collected.

Here is a summary of the *IQC* parameters that affect streaming news headlines:

Parameter	Data type	Default	Description
Symbol or Symbols ¹³⁰	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits the query to the specified symbols and meta-tags only (or to all symbols, if empty). Examples: <ul style="list-style-type: none"> 'IBM' 'IBM:AAPL:GOOG' 'IBM,AAPL,GOOG' {'IBM', 'AAPL', 'GOOG'} 'BZRatings:BZTradingIdeas'
Sources	colon or comma-delimited string or cell-array of strings	" (empty string), meaning all	Limits the query to the specified news sources only (or to all sources, if empty). Examples: <ul style="list-style-type: none"> 'DTN' 'DTN:CPR:BEN' 'DTN,CPR,BEN' {'DTN', 'CPR', 'BEN'}
NumOfEvents	integer	Inf	One of: <ul style="list-style-type: none"> inf – continuous endless streaming headlines for the specified security N>1 – stream only N headlines 1 – get only a single headline (default) 0 – stop streaming headlines -1 – return the latest accumulated headlines data while continuing to stream new headlines data
MaxItems	integer	Inf	Number of streaming headlines stored in a cyclic buffer. Once this number of headlines has been received, the oldest headline is discarded whenever a new headline arrives.
DataType	string	'headline'	Ignored – only headlines can be streamed
GetStory	logical (true/false)	false	If false (default), only store the incoming headline messages. If true or 1, automatically fetch and store the full story text for each incoming headline.

¹³⁰ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

8 Lookup of symbols and codes

A list of symbols and lookup codes that match a specified set of criteria can be retrieved using the 'lookup' and 'chain' actions. Various different lookups can be requested, which differ by the **DataType** parameter.

8.1 Symbols lookup

To retrieve a list of symbols that match certain criteria, set the action to 'lookup', **DataType** to 'symbols' and add one or more filtering criteria: **Name**, **Description**, **Market**, **SecType**, **SIC**, and/or **NAICS**:

```
>> data = IQC('lookup', 'DataType', 'symbols', 'Name', 'IBM')
data =
1086x1 struct array with fields:
    Symbol
    Description
    Market_ID
    Market_Name
    Sec_Type_ID
    Sec_Type
>> data(1)
ans =
    Symbol: 'IBM'
    Description: 'INTERNATIONAL BUSINESS MACHINE'
    Market_ID: 7
    Market_Name: 'New York Stock Exchange (NYSE)'
    Sec_Type_ID: 1
    Sec_Type: 'Equity'
>> data(2)
ans =
    Symbol: 'IBMG'
    Description: 'ISHARES IBONDS SEP 2018 MUNI BOND'
    Market_ID: 11
    Market_Name: 'NYSE Archipelago (NYSE_ARCA)'
    Sec_Type_ID: 1
    Sec_Type: 'Equity'
>> data(9)
ans =
    Symbol: 'IBM1804E120'
    Description: 'IBM MAY 2018 C 120.00'
    Market_ID: 14
    Market_Name: 'OPRA System'
    Sec_Type_ID: 2
    Sec_Type: 'Index/Equity Option'
>> data(end)
ans =
    Symbol: 'IBZ18-IBM19'
    Description: '30 DAY INTERBANK CASH RATE DEC 18/JUN 19'
    Market_ID: 64
    Market_Name: 'ASX24 Commodities Exchange (ASXCM)'
    Sec_Type_ID: 10
    Sec_Type: 'Future Spread'
```

IQFeed returns a list of symbols whose symbol name contains (not necessarily begins with) the term 'IBM', from different markets (exchanges) and different security types.

Note that the **Name** and **Description** filtering criteria are case-insensitive (so 'IBM', 'Ibm' and 'ibm' would all result in the same list of symbols), and also that they match their string value anywhere within the corresponding asset field.

You can narrow-down the results by entering more-specific parameter values (e.g. 'IBM180' rather than 'IBM'), or by specifying additional filtering parameters. For example, to filter the IBM list just to assets that include 'business' in their **Description**:

```
>> data = IQC('lookup', 'DataType','symbols', 'name','ibm', ...
               'Description','business')

data =
    8x1 struct array with fields:
        Symbol
        Description
        Market_ID
        Market_Name
        Sec_Type_ID
        Sec_Type

>> data = struct2table(data)
data =
    8x6 table

    Symbol      Description      Market_ID      Market_Name      Sec_Type_ID      Sec_Type
    _____
    'IBM'       'INTERNATIONAL BUSINESS MACHINE' 7      'New York Stock Exchange (NYSE)' 1      'Equity'
    'IBM19.CB'  'INTL BUSINESS MACHINES' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'IBM25.CB'  'INTL BUSINESS MACHINES' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'IBM27.CB'  'INTL BUSINESS MACHINES' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'IBM28.CB'  'INTL BUSINESS MACHINES' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'IBM39.CB'  'INTERNATIONAL BUSINESS MACHS SR NT 5.6%' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'IBM46.CB'  'INTERNATIONAL BUSINESS MACHINES CORP 4.7' 7      'New York Stock Exchange (NYSE)' 5      'Bond'
    'L.IBM'     'INTERNATIONAL BUSINESS MACHINES CORPORATION' 56     'London Stock Exchange (LSE)' 1      'Equity'
```

Unlike the **Name** and **Description** (which match strings), the **SIC** and **NAICS** parameters are numeric and match the *beginning* of the corresponding SIC/NAICS sector/industry code. For example, the following query returns all assets that have 'inc' in their **Description** and belong to any sector whose SIC code begins with 83:¹³¹

```
>> data = IQC('lookup', 'DataType','symbols', 'Description','inc', 'SIC',83)
data =
    6x1 struct array with fields:
        Symbol
        Description
        Market_ID
        Market_Name
        Sec_Type_ID
        Sec_Type
        SIC_ID
        SIC_Desc

>> data(1)
ans =
    Symbol: 'HQGE'
    Description: 'HQ GLOBAL ED INC'
    Market_ID: 3
    Market_Name: 'Nasdaq other OTC'
    Sec_Type_ID: 1
    Sec_Type: 'Equity'
    SIC_ID: 8331
    SIC_Desc: 'JOB TRAINING AND VOCATIONAL REHABILITATION SERVICES'

>> disp([data.Symbol; data.Description; data.SIC_ID; data.SIC_Desc])
'HQGE' 'HQ GLOBAL ED INC' [8331] 'JOB TRAINING AND ...'
'KVIL' 'KIDVILLE INC' [8351] 'CHILD DAY CARE SERVICES'
'DRWN' 'A CLEAN SLATE INC.' [8361] 'RESIDENTIAL CARE'
'NVOS' 'NOVO INTEGRATED SCIENCES INC...' [8361] 'RESIDENTIAL CARE'
'SPRV' 'SUPURVA HEALTHCARE GROUP INC...' [8361] 'RESIDENTIAL CARE'
```

¹³¹ In this example, the matching SIC codes were 8331 (HQGE), 8351 (KVIL), 8361 (DRWN, NVOS, SPRV). IQFeed has a bug (as of October 2019): no data is returned if SIC/NAICS < 10. (<http://forums.iqfeed.net/index.cfm?page=topic&topicID=5653>).

When you specify a **SIC** or **NAICS** filtering criteria, the result contains two additional fields (either `SIC_ID` and `SIC_Desc`, or `NAICS_ID` and `NAICS_Desc`, respectively), in addition to the standard fields (`Symbol`, `Description`, `Market_ID`, `Market_Name`, `Sec_Type_ID` and `Sec_Type`).¹³²

Note that it is possible that not all the requested symbols will be received before *IQC*'s timeout (default value: 5 secs) returns the results:¹³³

```
>> data = IQC('lookup', 'DataType','symbols', 'Name','GOOG')
Warning: IQC timeout: only partial data is returned. Perhaps the Timeout
parameter should be set to a value larger than 5
data =
3848x1 struct array with fields:
    Symbol
    Description
    Market_ID
    Market_Name
    Sec_Type_ID
    Sec_Type
```

To control the maximal duration that *IQC* will wait for the data, set the **Timeout** parameter. For example, to wait up to 30 secs to collect the complete list of symbols:

```
>> data = IQC('lookup', 'DataType','symbols', 'Name','GOOG', 'timeout',30)
data =
6812x1 struct array with fields:
...
```

Naturally, it is quite possible that no symbol is found that matches the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab's `struct2table()` or `struct2cell()` functions):

```
>> data = IQC('lookup', 'DataType','symbols', 'Description','inc', 'NAICS',83)
data =
[]

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

An error message will result if you try to specify both **SIC** and **NAICS** filtering criteria – only one (or none) of them is permitted in a lookup query:

```
>> data = IQC('lookup', 'DataType','symbols', 'NAICS',1234, 'SIC',83)
You can specify either SIC or NAICS parameter, but not both of them, in a
symbol lookup query
```

An error message will also result if you do not specify at least one of the filtering criteria **Name**, **Description**, **Market**, **SecType**, **SIC**, **NAICS**:

```
>> data = IQC('lookup', 'DataType','symbols')
At least one of Name, Description, Market, SecType, SIC or NAICS parameters
must be specified in a symbol lookup query
```

¹³² The description of the various numeric codes for `Market_ID`, `Sec_Type_ID`, `SIC` and `NAICS` can be fetched separately – see §8.3-§8.6 for details

¹³³ *IQC* can process ~1000 symbols per second; coupled with the network and server-processing latencies we can expect ~4000 symbols to accumulate before the default timeout of 5 seconds kicks in.

You can filter the results based on one or more markets, and/or security types, using the **Market** and **SecType** parameters (see §8.3, §8.4 for valid values). For example:

```
>> struct2table(IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'SecType', 'Equity'))
ans =
2x6 table
      Symbol      Description      Market_ID      Market_Name      Sec_Type_ID      Sec_Type
      -----
      'GOOG'      'ALPHABET INC CLASS C'      21      'Nasdaq Global Select Market (NGSM)'      1      'Equity'
      'GOOGL'      'ALPHABET INC CLASS A'      21      'Nasdaq Global Select Market (NGSM)'      1      'Equity'

>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'Market', 'NGSM');
```

Multiple **Markets** and/or **SecTypes**¹³⁴ can be specified using a cell array. For example, to get the list of all active (non-expired) GOOG equities and options:¹³⁵

```
>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', ...
              'SecTypes', {'Equity', 'IEOption'}, 'Timeout', 20)

data =
8056x1 struct array with fields:
      Symbol
      Description
      Market_ID
      Market_Name
      Sec_Type_ID
      Sec_Type
```

You can specify both **Market(s)** and **SecType(s)** to get an even more granular filtering. For example, to lookup only future options traded on CBOT:

```
>> data = IQC('lookup', 'datatype', 'symbols', 'name', symbol, ...
              'SecTypes', 'FOption', 'Markets', 'CBOT');
```

Similarly, to lookup VIX (volatility) futures and future-spreads (but not combined future volume OI symbols such as @VX1.OI.Z) on the CBOE Futures Exchange (CFE):

```
>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'vx', ...
              'SecTypes', {'Future', 'Spread'}, 'Markets', 'CFE');
```

If you specify one or more invalid **Market(s)** or **SecType(s)**, you will get an error. For example, a typical error is to specify a **SecType** of 'Option' instead of 'IEOption':

```
>> d = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'SecTypes', {'Equity', 'Option'})

Invalid SecType(s) "OPTION". Allowed values: ARGUS, ARGUSFC, BONDS, CALC,
COMBINED_FOPTION, COMBINED_FUTURE, COMM3, EQUITY, FAST_RACKS, FOPTION,
FOPTION_IV, FOREX, FORWARD, FUTURE, ICSPREAD, IEOPTION, INDEX, ISO, JACOBSEN,
MKTRPT, MKTSTATS, MONEY, MUTUAL, NP_CAPACITY, NP_FLOW, NP_POWER,
PETROCHEMWIRE, PRECMTL, RACKS, RFSPOT, SNL_ELEC, SNL_NG, SPOT, SPREAD,
STRATSPREAD, STRIP, SWAPS, TREASURIES
```

Instead of **Market** name(s) or **SecType** name(s), you can specify their corresponding numeric codes,¹³⁶ as a scalar integer value or as a numeric array of integers:

```
>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'SecTypes', 1);
>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'SecTypes', [1, 2]);

>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'Markets', 21);
>> data = IQC('lookup', 'datatype', 'symbols', 'name', 'GOOG', 'Markets', [21, 14]);
```

¹³⁴ Note that you can use either **Market** or **Markets** as the parameter name, and similarly, either **SecType** or **SecTypes**.

¹³⁵ IQFeed only returns the symbols of active (non-expired) options/futures. See §8.2 for details about expired contracts.

¹³⁶ See §8.3 and §8.4 for the list of numeric codes that correspond to each market and security type

Here is a summary of the *IQC* parameters that affect symbols lookup:

Parameter	Data type	Default	Description
Name	string	" (empty string)	Limits the query to assets that contain the specified string in their symbol name (case insensitive, <i>anywhere</i> within the symbol name)
Description	string	" (empty string)	Limits the query to assets that contain the specified string in their description (case insensitive, <i>anywhere</i> within the description)
Market or Markets ¹³⁷	integer, numeric array, string, or cell-array of strings	[] (empty)	Limits the query to assets that belong to the specified market code(s) (scalar integer or numeric array), or market name(s) (case-insensitive string or cell-array of strings). See §8.3 for details on valid values.
SecType or SecTypes ¹³⁸	integer, numeric array, string, or cell-array of strings	[] (empty)	Limits the query to assets that have the specified security type code(s) (scalar integer or numeric array), or security type name(s) (case-insensitive string or cell-array of strings). See §8.4 for details on valid values.
SIC	integer	[] (empty)	Limits the query to assets that belong to the specified SIC sector/industry (matches the <i>beginning</i> of the SIC number) See §8.5 for details on valid values.
NAICS	integer	[] (empty)	Limits the query to assets that belong to the specified NAICS sector/industry (matches the <i>beginning</i> of the NAICS number) See §8.6 for details on valid values.
Timeout	number	5.0	Max # of seconds to wait for incoming data (0-9000, where 0 means infinite)

¹³⁷ In *IQC*, the **Market** and **Markets** parameters are synonymous – you can use either of them, in any capitalization

¹³⁸ In *IQC*, the **SecType** and **SecTypes** parameters are synonymous – you can use either of them, in any capitalization

8.2 Options/futures chain

To retrieve a list of symbols that belong to a certain options/futures chain and match certain criteria, set the action to 'chain'; **DataType** to one of 'options' (default), 'futures', 'foptions' (future options), or 'spreads'; **Symbol** to the underlying contract's symbol; and then add optional filtering criteria. For example:¹³⁹

```
>> symbols = IQC('chain', 'Symbol', 'GOOG') % options chain for GOOG
symbols =
1x1454 cell array
Columns 1 through 4
    'GOOG1803H1000'    'GOOG1803H1010'    'GOOG1803H1020'    'GOOG1803H1030'
Columns 5 through 8
    'GOOG1803H1040'    'GOOG1803H1050'    'GOOG1803H1055'    'GOOG1803H1060'
Columns 9 through 12
    'GOOG1803H1065'    'GOOG1803H1070'    'GOOG1803H1075'    'GOOG1803H1077.5'
...
```

Depending on **DataType**, several filtering parameters can be specified: All chain queries support the **Symbol**, **Months**, and **NearMonths** parameters. Options-related queries (**DataType**='options' or 'foptions') also support a **Side** parameter. Index/equity options query (**DataType**='options') also supports **IncludeBinary**, **MinStrike**, **MaxStrike**, **NumInMoney**, **NumOutOfMoney** parameters. All chain queries *except* 'options' also support the **Years** parameter. Here's an example filtered chain query:

```
% Report GOOG options having strike price between $1000-$1010 in next 4 months
>> symbols = IQC('chain', 'symbol', 'GOOG', 'NearMonths', 4, ...
    'MinStrike', 1000, 'MaxStrike', 1010)

symbols =
1x58 cell array
Columns 1 through 4
    'GOOG1803H1000'    'GOOG1803H1010'    'GOOG1810H1000'    'GOOG1810H1005'
Columns 5 through 8
    'GOOG1810H1010'    'GOOG1813G1000'    'GOOG1813G1002.5'    'GOOG1813G1005'
Columns 9 through 12
    'GOOG1813G1007.5'    'GOOG1813G1010'    'GOOG1817H1000'    'GOOG1817H1005'
...
```

The following table lists the valid combinations of filtering parameters per **DataType**:

↓parameter \ DataType : →	options	future	spread	foption
Symbol	ok	ok	ok	ok
Side	ok	N/A	N/A	ok
Months	ok	ok	ok	ok
NearMonths	ok	ok	ok	ok
Years	N/A	ok	ok	ok
IncludeBinary	ok	N/A	N/A	N/A
MinStrike	ok	N/A	N/A	N/A
MaxStrike	ok	N/A	N/A	N/A
NumInMoney	ok	N/A	N/A	N/A
NumOutOfMoney	ok	N/A	N/A	N/A

¹³⁹ The option contract names in IQFeed use a variant of the OPRA OSI format. See

<http://www.iqfeed.net/symbolguide/index.cfm?symbolguide=guide&displayaction=support%C2%A7ion=guide&web=iqfeed&guide=options&web=IQFeed&type=stock>. Note that the name might change when corporate actions (such as splits) occur, for example: BBD1918A15 vs. BBD11918A15.45 (<http://forums.iqfeed.net/index.cfm?page=topic&topicID=5495>).

Note that if you filter by **MinStrike** and/or **MaxStrike**, you cannot also filter by **NumInMoney/ NumOutOfMoney** (and vice versa):

```
>> IQC('chain', 'symbol', 'FB', 'NumInMoney', 2, 'NumOutOfMoney', 2, 'MinStrike', 90)
You cannot specify both a strike range and number of contracts in/out of money
in 'chain' query - choose only one set
```

Similarly, you can only specify one of the **Months**, **NearMonths** parameters, not both:

```
>> IQC('chain', 'symbol', 'FB', 'Months', 2:6, 'NearMonths', 3)
Either the Months or the NearMonths parameter can be specified, but not both,
in a 'chain' query
```

If no symbols match the specified criteria, or if you do not have the necessary market permissions (subscription), then the *IQC* query will return an empty cell array:

```
>> symbols = IQC('chain', 'datatype', 'spreads', 'symbol', 'C', 'years', 2010:2019)
symbols =
0x0 empty cell array
```



Note: IQFeed only returns active (non-expired) contracts. A[huge] static text file containing a [very long] list of expired symbols is available for download¹⁴⁰ and starting with IQFeed client 6.1 you can also fetch this data using *IQC* (see §5.6).

If you set **WhatToShow** to 'quotes', you'll receive an array of structs that contain the corresponding latest (top-of-market) quotes data for the corresponding symbols:

```
>> data = IQC('chain', 'symbol', 'GOOG', 'NearMonths', 4, ...
              'MinStrike', 1000, 'MaxStrike', 1010, ...
              'WhatToShow', 'quotes')

data =
58x1 struct array with fields:
    Symbol
    ...

>> data(1)
ans =
struct with fields:
    Symbol: 'GOOG1803H1000'
    Most_Recent_Trade: 120
    Most_Recent_Trade_Size: 1
    Most_Recent_Trade_Time: '15:57:12.930497'
    Total_Volume: 0
    Bid: 140.5
    Bid_Size: 3
    Ask: 150.1
    Ask_Size: 1
    ...
    Close: 120
    Message_Contents: 'Cbacv'
    Message_Description: ...
    Most_Recent_Trade_Conditions: 1
    Trade_Conditions_Description: 'Normal Trade'
    Most_Recent_Market_Name: 'MIAX PEARL Options exchange'

>> symbols = {data.Symbol}
symbols =
1x58 cell array
Columns 1 through 4
    'GOOG1803H1000'    'GOOG1803H1010'    'GOOG1810H1000'    'GOOG1810H1005'
Columns 5 through 8
    'GOOG1810H1010'    'GOOG1813G1000'    'GOOG1813G1002.5'    'GOOG1813G1005'
...
```

¹⁴⁰ <http://www.dtniq.com/beta/IEOPTION.zip>. See <http://forums.iqfeed.net/index.cfm?page=topic&topicID=3326> for details.

Note: if you request quotes for multiple chain symbols, especially if you set **UseParallel** to **true**, you might reach your IQFeed account's symbols-limit (`MaxSymbols`; see §9.3). In such cases, IQFeed-generated error messages will be displayed on the Matlab console:

```
Level1 symbol limit reached - symbol 'GOOG2019R600' not serviced!
```

Also note that the returned quotes data is subject to the **Fields** parameter value that was set in the most recent quotes data query (see §4.2).

Also note that some of these structs (especially for out-of-money contracts) may contain empty/invalid data, since their corresponding contract was never traded. For example:

```
>> data(7)
ans =
    struct with fields:
        Symbol: 'GOOG1813G1002.5'
        Most_Recent_Trade: []
        Most_Recent_Trade_Size: []
        Most_Recent_Trade_Time: []
        Most_Recent_Trade_Market_Center: []
        Total_Volume: 0
        Bid: 133.4
        Bid_Size: 2
        Ask: 140.2
        Ask_Size: 1
        Open: []
        High: []
        Low: []
        Close: []
        Message_Contents: 'bav'
        Message_Description: 'A bid update occurred; An ask update
occurred; A volume update occurred'
        Most_Recent_Trade_Conditions: 1
        Trade_Conditions_Description: 'Normal Trade'
        Most_Recent_Market_Name: ''
```

For this reason, you should be careful when concatenating the struct array's data into numeric arrays. In this example, only 40 of the 58 contracts had a Close price, so concatenating into a numeric array results in an array that only has 40 data items:

```
>> [data.Close]
ans =
    Columns 1 through 8
        120    130.7    140.67    131.99    150.1    138.8    139.5    99.47
    Columns 9 through 16
        103.28    130.9    179.5    137.5    190.17    89.3    145    3.84
    Columns 17 through 24
         6         7.5         5.3         7.14         0.3         0.3         1.1         1.32
    Columns 25 through 32
         1.05         5.56         9.9         6.35         0.67         0.75         1.23         10
    Columns 33 through 40
        15.43    16.33    27.21    32.3    33.4    6.49    2.5    3.37
```

...instead, it is better in most cases to use cell arrays, where we can see empty cells:

```
>> {data.Close}
ans =
    1x58 cell array
    Columns 1 through 8
        [120]    []    [130.7]    []    []    [140.67]    []    []
    Columns 9 through 16
        []    []    [131.99]    [150.1]    [138.8]    [139.5]    []    [99.47]
    Columns 17 through 24
        []    [103.28]    [130.9]    [179.5]    [137.5]    [190.17]    []    [89.3]
    Columns 25 through 33
        ...
```

Similarly, set **WhatToShow='fundamental'** to get the fundamental data for all symbols in the requested chain. For example:

```
>> data = IQC('chain', 'symbol', 'GOOG', 'NearMonths', 4, ...
              'MinStrike', 1000, 'MaxStrike', 1010, ...
              'WhatToShow', 'fundamental')

data =
58x1 struct array with fields:
    Symbol
    Exchange_ID
    PE
    Average_Volume
    x52_Week_High
    x52_Week_Low
    Calendar_Year_High
    ...

>> data(1)
ans =
struct with fields:
    Symbol: 'GOOG1803H1000'
    Exchange_ID: 'E'
    PE: []
    Average_Volume: []
    x52_Week_High: 120
    x52_Week_Low: 120
    Calendar_Year_High: []
    Calendar_Year_Low: []
    ...
    Fiscal_Year_End: []
    Company_Name: 'GOOG AUG 2018 C 1000.00'
    ...
    Expiration_Date: '08/03/2018'
    Strike_Price: 1000
    NAICS: []
    Exchange_Root: []
    Option_Premium_Multiplier: 100
    Option_Multiple_Deliverable: 0
    Price_Format_Description: 'Two decimal places'
    Exchange_Description: 'Euronext Index Derivatives (ENID)'
    Listed_Market_Description: 'OPRA System'
    Security_Type_Description: 'Index/Equity Option'
    SIC_Description: ''
    NAICS_Description: ''

>> [data.Strike_Price]
ans =
Columns 1 through 8
    1000    1010    1000    1005    1010    1000    1002.5    1005
Columns 9 through 16
    1007.5    1010    1000    1005    1010    1000    1002.5    1005
Columns 17 through 24
    1007.5    1010    1000    1005    1010    1000    1005    1010
Columns 25 through 32
    1000    1005    1010    1000    1010    1000    1010    1000
Columns 33 through 40
    1005    1010    1000    1002.5    1005    1007.5    1010    1000
...
```

Here is a summary of the *IQC* parameters that affect chain symbols lookup:

Parameter	Data type	Default	Description
Symbol	string	" must be set!	Symbol name of the underlying contract. This is a mandatory parameter – it must be set. Note: Multiple symbols are NOT supported.

DataType	string	'options'	One of: <ul style="list-style-type: none"> 'options' (default) – on index/equity 'future' 'spread' – future calendar spreads 'foptions' – options on future
Side	string	'cp' (meaning both calls and puts)	One of: <ul style="list-style-type: none"> 'cp' (default) – both calls and puts 'c' – calls only 'p' – puts only Only relevant if DataType ='options'/'foptions'
WhatToShow	string	'symbols'	One of: <ul style="list-style-type: none"> 'symbols' (default) – list of symbols in chain 'quotes' – return the latest quotes data 'fundamental' – return fundamental data
Months	various	[] meaning all	One of: <ul style="list-style-type: none"> Numeric month value(s) between 1-12 (e.g.: 4, 2:5, [1,4,7]) English month name (e.g. 'August', 'Apr') English month names in cell array (e.g. {'Apr', 'July', 'September', 'Dec'}) Financial month codes from the list FGHIJKMNQUVXZ (e.g. 'JKMN') Cannot be specified together with NearMonths
NearMonths	integer (0-99)	[]	Number of nearby contract months to report. ¹⁴¹ Cannot be specified together with Months .
Years	integer scalar/array	[] meaning current year	1+ years e.g. 2013:2019, default: current year. Only relevant when DataType ≠ 'options'.
IncludeBinary	logical	false or 0	If true, binary/weekly options ¹⁴² are reported, otherwise (default) they are not. This parameter is only relevant when DataType ='options'.
MinStrike	number	[]	Only report options having a higher strike price; only relevant when DataType ='options'.
MaxStrike	number	[]	Only report options having a lower strike price; only relevant when DataType ='options'.
NumInMoney	integer	[]	Only report this number of options in the money; only relevant if DataType ='options'.
NumOutOfMoney	integer	[]	Only report this number of options out of money; only relevant if DataType ='options'.
UseParallel	logical (true/false)	false	If set to true or 1, then querying chain quotes will be done in parallel if possible (see §3.6).
MaxWorkers	integer	(the current pool size)	Max number of parallel workers to use (up to the current pool size) when UseParallel =1



Note: Options/futures chain lookup is only available in the Professional *IQC* license.

¹⁴¹ IQFeed officially supports only 0-4, but in practice higher values are accepted, reporting contracts that expire farther out in the future (for example, 2.5 years for SPX). Note: this is undocumented IQFeed behavior, so specifying a value of 5 or higher may possibly not work properly (or at all) in certain cases. See <http://forums.iqfeed.net/index.cfm?page=topic&topicID=5508>

¹⁴² Weekly options are only excluded with IQFeed client 6.1 or newer; binary options are excluded with all clients.

8.3 Markets lookup

To retrieve a list of markets (exchanges), set the action to 'lookup' and **DataType** to 'markets':

```
>> data = IQC('lookup', 'DataType', 'markets')
data =
    474x1 struct array with fields:
        id
        name
        description
        groupId
        groupName

>> data(1)
ans =
        id: 1
       name: 'NGM'
description: 'Nasdaq Global Market'
   groupId: 5
  groupName: 'NASDAQ'

>> data(2)
ans =
        id: 2
       name: 'NCM'
description: 'National Capital Market'
   groupId: 5
  groupName: 'NASDAQ'
```

You can convert the data into a [perhaps] more readable form using Matlab's builtin `struct2cell()` and `struct2table()` functions:

```
>> struct2cell(data)
ans =
    9x5 cell array
    [1] 'NGM'          'Nasdaq Global Market'          [5] 'NASDAQ'
    [2] 'NCM'          'National Capital Market'        [5] 'NASDAQ'
    [3] 'OTC'          'Nasdaq other OTC'              [5] 'NASDAQ'
    [4] 'OTCBB'        'Nasdaq OTC Bulletin Board'      [5] 'NASDAQ'
    [5] 'NASDAQ'       'Nasdaq'                        [5] 'NASDAQ'
    [6] 'NYSE_AMERICAN' 'NYSE American (Equities and Bonds)' [6] 'NYSE_AMERICAN'
    [7] 'NYSE'         'New York Stock Exchange'        [7] 'NYSE'
    [8] 'CHX'          'Chicago Stock Exchange'         [0] 'NONE'
    [9] 'PHLX'         'Philadelphia Stock Exchange'    [0] 'NONE'
    ...

>> struct2table(data)
ans =
    9x5 table
        id      name      description      groupId      groupName
    _____
    1 'NGM'      'Nasdaq Global Market'      5      'NASDAQ'
    2 'NCM'      'National Capital Market'    5      'NASDAQ'
    3 'OTC'      'Nasdaq other OTC'          5      'NASDAQ'
    4 'OTCBB'    'Nasdaq OTC Bulletin Board'  5      'NASDAQ'
    5 'NASDAQ'   'Nasdaq'                    5      'NASDAQ'
    6 'NYSE_AMERICAN' 'NYSE American (Equities and Bonds)' 6      'NYSE_AMERICAN'
    7 'NYSE'     'New York Stock Exchange'    7      'NYSE'
    8 'CHX'      'Chicago Stock Exchange'     0      'NONE'
    9 'PHLX'     'Philadelphia Stock Exchange' 0      'NONE'
```

You can narrow-down the results by specifying **ID**, **Name** and/or **Description** filtering parameters. For example, to list only markets whose **Description** contains 'Nasdaq':

```
>> data = IQC('lookup', 'DataType','markets', 'Description','Nasdaq')
data =
    10x1 struct array with fields:
        id
        name
        description
        groupId
        groupName
>> disp(struct2cell(data)')
[ 1] 'NGM'      'Nasdaq Global Market'      [ 5] 'NASDAQ'
[ 3] 'OTC'      'Nasdaq other OTC'      [ 5] 'NASDAQ'
[ 4] 'OTCBB'    'Nasdaq OTC Bulletin Board' [ 5] 'NASDAQ'
[ 5] 'NASDAQ'   'Nasdaq'      [ 5] 'NASDAQ'
[15] 'NASD_ADF' 'Nasdaq Alternate Display facility' [ 5] 'NASDAQ'
[19] 'NTRF'     'Nasdaq Trade Reporting Facility' [ 5] 'NASDAQ'
[21] 'NGSM'     'Nasdaq Global Select Market' [ 5] 'NASDAQ'
[105] 'PK_NASDAQ' 'Pink Sheets - NASDAQ Listed' [ 90] 'PK_SHEETS'
[134] 'N2EX'     'NASDAQ OMX-Nord Pool' [134] 'N2EX'
[139] 'NFX'      'NASDAQ OMX Futures' [139] 'NFX'
```

Naturally, it is quite possible that no markets exist that match the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab's `struct2table()` or `struct2cell()` functions):

```
>> data = IQC('lookup', 'DataType','markets', 'Name','xyz')
data =
    []
>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

Here is a summary of the *IQC* parameters that affect markets lookup:

Parameter	Data type	Default	Description
ID	integer or numeric array	[] (empty)	Limits the query to the specified ID(s).
Name	string	" (empty string)	Limits the query to markets that contain the specified string in their name or groupName (case insensitive, <i>anywhere</i> within the name)
Description	string	" (empty string)	Limits the query to markets that contain the specified string in their description (case insensitive, <i>anywhere</i> within the description)

Note: IQFeed does not currently offer additional information about the listed markets, such as their time-zone, operating times, and duration for delayed quotes.¹⁴³ A table of time-zones and delay amounts (but not operating times) for markets in DTN's ProphetX service is posted on <http://pxweb.dtn.com/PXWebDoc/pages/Markets.aspx>. These markets generally overlap IQFeed's markets, but the market names are somewhat different (for example, Singapore's International Monetary Exchange is named 'SIME' in ProphetX vs. 'SGX' in IQFeed). It is unclear to what extent ProphetX's market information is up-to-date or relevant to IQFeed, so be careful when using it.

¹⁴³ <http://forums.dtn.com/index.cfm?page=topic&topicID=5740>

8.4 Security types lookup

To retrieve a list of security types, set action to 'lookup' and **DataType** to 'sectypes':

```
>> data = IQC('lookup', 'DataType', 'sectypes')
data =
    38x1 struct array with fields:
        id
        name

>> data(1)
ans =
        id: 1
       name: 'EQUITY'
  description: 'Equity'

>> data(2)
ans =
        id: 2
       name: 'IEOPTION'
  description: 'Index/Equity Option'
```

You can convert the data into a [perhaps] more readable form using Matlab's builtin `struct2cell()` and `struct2table()` functions:

```
>> disp(struct2cell(data)')
[ 1] 'EQUITY'      'Equity'
[ 2] 'IEOPTION'    'Index/Equity Option'
[ 3] 'MUTUAL'      'Mutual Fund'
[ 4] 'MONEY'       'Money Market Fund'
[ 5] 'BONDS'       'Bond'
[ 6] 'INDEX'       'Index'
[ 7] 'MKTSTATS'    'Market Statistic'
[ 8] 'FUTURE'      'Future'
[ 9] 'FOPTION'     'Future Option'
[10] 'SPREAD'      'Future Spread'
[11] 'SPOT'        'Spot'
[12] 'FORWARD'     'Forward'
[13] 'CALC'        'DTN Calculated Statistic'
[14] 'STRIP'       'Calculated Future Strip'
[16] 'FOREX'       'Foreign Monetary Exchange'
[17] 'ARGUS'       'Argus Energy'
[18] 'PRECMTL'     'Precious Metals'
[19] 'RACKS'       'Racks Energy'
[20] 'RFSPOT'      'Refined Fuel Spot'
[21] 'ICSPREAD'    'Inter-Commodity Future Spread'
[22] 'STRATSPREAD' 'Strategy Spread'
[23] 'TREASURIES'  'Treasuries'
[24] 'SWAPS'       'Interest Rate Swap'
[25] 'MKTRPT'      'Market Reports'
[26] 'SNL_NG'      'SNL Natural Gas'
[27] 'SNL_ELEC'    'SNL Electricity'
[28] 'NP_CAPACITY' 'Nord Pool-N2EX Capacity'
[29] 'NP_FLOW'     'Nord Pool-N2EX Flow'
[30] 'NP_POWER'    'Nord Pool-N2EX Power Prices'
[31] 'COMM3'       'Commodity 3'
[32] 'JACOBSEN'    'The Jacobsen'
[33] 'ISO'         'Independent Systems Operator Data (Genscape)'
[34] 'FAST_RACKS'  'Fast Racks (Racks On Wheels)'
[35] 'COMBINED_FUTURE' 'Combined Future Volume OI'
[36] 'COMBINED_FOPTION' 'Combined FOption Volume OI'
[37] 'ARGUSFC'     'Argus Forward Curve'
[38] 'PETROCHEMWIRE' 'PetroChemWire'
[39] 'FOPTION_IV'  'FOption Implied Volatility'
```

```
>> disp(struct2table(data))
```

id	name	description
1	'EQUITY'	'Equity'
2	'IEOPTION'	'Index/Equity Option'
3	'MUTUAL'	'Mutual Fund'
4	'MONEY'	'Money Market Fund'
5	'BONDS'	'Bond'
6	'INDEX'	'Index'
7	'MKTSTATS'	'Market Statistic'
8	'FUTURE'	'Future'
9	'FOPTION'	'Future Option'
10	'SPREAD'	'Future Spread'
11	'SPOT'	'Spot'
12	'FORWARD'	'Forward'
...		

You can narrow-down the results by specifying **ID**, **Name** and/or **Description** filtering parameters. For example, to list only secTypes whose **Description** contains 'Option':

```
>> struct2table(IQC('lookup', 'DataType', 'sectypes', 'Description', 'option'))
ans =
4x3 table
    id      name      description
    ---  ---  ---
    2    'IEOPTION'    'Index/Equity Option'
    9    'FOPTION'     'Future Option'
   36    'COMBINED_FOPTION' 'Combined FOption Volume OI'
   39    'FOPTION_IV'   'FOption Implied Volatility'
```

Naturally, it is quite possible that no security types exist that match the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab's `struct2table()` or `struct2cell()` functions):

```
>> data = IQC('lookup', 'DataType', 'sectypes', 'Name', 'xyz')
data =
[]

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

Here is a summary of the *IQC* parameters that affect security types lookup:

Parameter	Data type	Default	Description
ID	integer or numeric array	[] (empty)	Limits the query to the specified ID(s).
Name	string	" (empty string)	Limits the query to secTypes that contain the specified string in their name (case insensitive, <i>anywhere</i> within the name)
Description	string	" (empty string)	Limits the query to secTypes that contain the specified string in their description (case insensitive, <i>anywhere</i> within the description)

8.5 SIC codes lookup

To retrieve a list of SIC sectors/industries, set action to 'lookup' and **Data Type** to 'SIC':

```
>> data = IQC('lookup', 'DataType', 'SIC')
data =
    1009x1 struct array with fields:
         id
    description

>> data(1)
ans =
         id: 111
    description: 'WHEAT'

>> data(2)
ans =
         id: 112
    description: 'RICE'
```

You can convert the data into a [perhaps] more readable form using Matlab's builtin `struct2cell()` and `struct2table()` functions:

```
>> disp(struct2cell(data)')

[111]    'WHEAT'
[112]    'RICE'
[115]    'CORN'
[116]    'SOYBEANS'
[119]    'CASH GRAINS, NOT ELSEWHERE CLASSIFIED'
[131]    'COTTON'
[132]    'TOBACCO'
[133]    'SUGARCANE AND SUGAR BEETS'
[134]    'IRISH POTATOES'
[139]    'FIELD CROPS, EXCEPT CASH GRAINS, NOT ELSEWHERE CLASSIFIED'
[161]    'VEGETABLES AND MELONS'
[171]    'BERRY CROPS'
[172]    'GRAPES'
[173]    'TREE NUTS'
[174]    'CITRUS FRUITS'
[175]    'DECIDUOUS TREE FRUITS'
[179]    'FRUITS AND TREE NUTS, NOT ELSEWHERE CLASSIFIED'
...

>> disp(struct2table(data))

    id    description
    ---    ---
    111    'WHEAT'
    112    'RICE'
    115    'CORN'
    116    'SOYBEANS'
    119    'CASH GRAINS, NOT ELSEWHERE CLASSIFIED'
    131    'COTTON'
    132    'TOBACCO'
    133    'SUGARCANE AND SUGAR BEETS'
    134    'IRISH POTATOES'
    139    'FIELD CROPS, EXCEPT CASH GRAINS, NOT ELSEWHERE CLASSIFIED'
    161    'VEGETABLES AND MELONS'
    171    'BERRY CROPS'
    172    'GRAPES'
    173    'TREE NUTS'
    174    'CITRUS FRUITS'
    175    'DECIDUOUS TREE FRUITS'
    179    'FRUITS AND TREE NUTS, NOT ELSEWHERE CLASSIFIED'
    ...
```

You can narrow-down the results by specifying **ID** or **Description** filtering parameters. For example, to list only the SIC codes whose **Description** contains ‘Oil’:

```
>> struct2table(IQC('lookup', 'DataType','SIC', 'Description','oil'))
ans =
    22x2 table
      id      description
    _____
    251  'BROILER, FRYER, AND ROASTER CHICKENS'
    711  'SOIL PREPARATION SERVICES'
   1381  'DRILLING OIL AND GAS WELLS'
   1382  'OIL AND GAS FIELD EXPLORATION SERVICES'
   1389  'OIL AND GAS FIELD SERVICES, NOT ELSEWHERE CLASSIFIED'
   2074  'COTTONSEED OIL MILLS'
   2075  'SOYBEAN OIL MILLS'
   2076  'VEGETABLE OIL MILLS, EXCEPT CORN, COTTONSEED, AND SOYBEAN'
   2077  'ANIMAL AND MARINE FATS AND OILS'
   2079  'SHORTENING, TABLE OILS, MARGARINE, AND OTHER EDIBLE FATS AND OILS'
   2673  'PLASTICS, FOIL, AND COATED PAPER BAGS'
   2843  'SURFACE ACTIVE AGENTS, FINISHING AGENTS, SULFONATED OILS, AND ASS'
   2844  'PERFUMES, COSMETICS, AND OTHER TOILET PREPARATIONS'
   2992  'LUBRICATING OILS AND GREASES'
   3353  'ALUMINUM SHEET, PLATE, AND FOIL'
   3443  'FABRICATED PLATE WORK (BOILER SHOPS)'
   3497  'METAL FOIL AND LEAF'
   3532  'MINING MACHINERY AND EQUIPMENT, EXCEPT OIL AND GAS FIELD MACHINER'
   3533  'OIL AND GAS FIELD MACHINERY AND EQUIPMENT'
   3677  'ELECTRONIC COILS, TRANSFORMERS, AND OTHER INDUCTORS'
   5983  'FUEL OIL DEALERS'
   6792  'OIL ROYALTY TRADERS'
```

Naturally, it is quite possible that no security types exist that match the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab’s `struct2table()` or `struct2cell()` functions):

```
>> data = IQC('lookup', 'DataType','SIC', 'Description','xyz')
data =
    []

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

Here is a summary of the *IQC* parameters that affect SIC codes lookup:

Parameter	Data type	Default	Description
ID	integer or numeric array	[] (empty)	Limits the query to the specified ID(s).
Description	string	" (empty string)	Limits the query to SIC entries that contain the specified string in their description (case insensitive, <i>anywhere</i> within the description)



Note: IQFeed has a confirmed internal bug as of October 2019: some ~150 SIC codes are not reported, although they have corresponding symbols and are reported by the symbols lookup query (§8.1).¹⁴⁴ Symbols having such SIC codes will have an empty `SIC_Description` field in the fundamental data query (§4.2) and empty `SIC_Desc` field in the symbols lookup query (§8.1).

¹⁴⁴ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=5653>

8.6 NAICS codes lookup

To retrieve a list of NAICS sectors/industries, set the action to 'lookup' and **DataType** to 'NAICS':

```
>> data = IQC('lookup', 'DataType', 'NAICS')
data =
    1175x1 struct array with fields:
         id
    description
>> data(1)
ans =
         id: 111110
    description: 'Soybean Farming'
>> data(2)
ans =
         id: 111120
    description: 'Oilseed (except Soybean) Farming'
```

You can convert the data into a [perhaps] more readable form using Matlab's builtin `struct2cell()` and `struct2table()` functions:

```
>> disp(struct2cell(data)')

[111110]    'Soybean Farming'
[111120]    'Oilseed (except Soybean) Farming'
[111130]    'Dry Pea and Bean Farming'
[111140]    'Wheat Farming'
[111150]    'Corn Farming'
[111160]    'Rice Farming'
[111191]    'Oilseed and Grain Combination Farming'
[111199]    'All Other Grain Farming'
[111211]    'Potato Farming'
[111219]    'Other Vegetable (except Potato) and Melon Farming'
[111310]    'Orange Groves'
[111320]    'Citrus (except Orange) Groves'
[111331]    'Apple Orchards'
[111332]    'Grape Vineyards'
[111333]    'Strawberry Farming'
[111334]    'Berry (except Strawberry) Farming'
[111335]    'Tree Nut Farming'
...
>> disp(struct2table(data))

      id      description
      _____
111110    'Soybean Farming'
111120    'Oilseed (except Soybean) Farming'
111130    'Dry Pea and Bean Farming'
111140    'Wheat Farming'
111150    'Corn Farming'
111160    'Rice Farming'
111191    'Oilseed and Grain Combination Farming'
111199    'All Other Grain Farming'
111211    'Potato Farming'
111219    'Other Vegetable (except Potato) and Melon Farming'
111310    'Orange Groves'
111320    'Citrus (except Orange) Groves'
111331    'Apple Orchards'
111332    'Grape Vineyards'
111333    'Strawberry Farming'
111334    'Berry (except Strawberry) Farming'
111335    'Tree Nut Farming'
...
```

You can narrow-down the results by specifying the **ID** or **Description** filtering parameter. For example, to list only NAICS codes whose **Description** contains ‘Oil’:

```
>> struct2table(IQC('lookup', 'DataType', 'NAICS', 'Description', 'oil'))
ans =
20x2 table
      id      description
-----
111120 'Oilseed (except Soybean) Farming'
111191 'Oilseed and Grain Combination Farming'
112320 'Broilers and Other Meat Type Chicken Production'
115112 'Soil Preparation, Planting, and Cultivating'
213111 'Drilling Oil and Gas Wells'
213112 'Support Activities for Oil and Gas Operations'
237120 'Oil and Gas Pipeline and Related Structures Construction'
311223 'Other Oilseed Processing'
311225 'Fats and Oils Refining and Blending'
322225 'Laminated Aluminum Foil Manufacturing for Flexible Packaging Uses'
324191 'Petroleum Lubricating Oil and Grease Manufacturing'
325620 'Toilet Preparation Manufacturing'
331315 ' Aluminum Sheet, Plate, and Foil Manufacturing' 145
332410 'Power Boiler and Heat Exchanger Manufacturing'
333132 'Oil and Gas Field Machinery and Equipment Manufacturing'
334416 'Electronic Coil, Transformer, and Other Inductor Manufacturing'
423810 'Construction and Mining (except Oil Well) Machinery and Equipment...'
454311 'Heating Oil Dealers'
486110 'Pipeline Transportation of Crude Oil'
811191 'Automotive Oil Change and Lubrication Shops'
```

Naturally, it is quite possible that no security types exist that match the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab’s `struct2table()` or `struct2cell()` functions):

```
>> data = IQC('lookup', 'DataType', 'NAICS', 'Description', 'xyz')
data =
[]

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

Here is a summary of the *IQC* parameters that affect NAICS codes lookup:

Parameter	Data type	Default	Description
ID	integer or numeric array	[] (empty)	Limits the query to the specified ID(s).
Description	string	" (empty string)	Limits the query to NAICS entries that contain the specified string in their description (case insensitive, <i>anywhere</i> within the description)



Note: IQFeed has a confirmed internal bug as of October 2019: some ~150 SIC codes are not reported, although they have corresponding symbols and are reported by the symbols lookup query (§8.1).¹⁴⁶ A similar bug also applies to NAICS.

¹⁴⁵ The extra space at the beginning of the description here is a typo in IQFeed’s data

¹⁴⁶ <http://forums.iqfeed.net/index.cfm?page=topic&topicID=5653>

8.7 Trade condition codes lookup

To retrieve a list of trade condition codes, set the action to 'lookup' and **DataType** to 'conditions':

```
>> data = IQC('lookup', 'DataType', 'conditions')
data =
    155x1 struct array with fields:
        id
        name
        description
>> data(1)
ans =
        id: 1
        name: 'REGULAR'
        description: 'Normal Trade'
>> data(2)
ans =
        id: 2
        name: 'ACQ'
        description: 'Acquisition'
```

You can convert the data into a [perhaps] more readable form using Matlab's builtin `struct2cell()` and `struct2table()` functions:

```
>> disp(struct2cell(data))

[ 1]    'REGULAR'    'Normal Trade'
[ 2]    'ACQ'       'Acquisition'
[ 3]    'CASHM'     'Cash Only Market'
[ 4]    'BUNCHED'   'Bunched Trade'
[ 5]    'AVGPRI'    'Average Price Trade'
[ 6]    'CASH'      'Cash Trade (same day clearing)'
[ 7]    'DIST'      'Distribution'
[ 8]    'NEXTDAY'   'Next Day Market'
[ 9]    'BURSTBSKT' 'Burst Basket Execution'
[10]    'BUNCHEDSOLD' 'Bunched Sold Trade'
[11]    'ORDETAIL'  'Opening/Reopening Trade Detail'
[12]    'INTERDAY'  'Intraday Trade Detail'
[13]    'BSKTONCLOSE' 'Basket Index on Close'
[14]    'RULE127'   'Rule - 127 Trade NYSE'
[15]    'RULE155'   'Rule - 155 Trade AMEX'
[16]    'SOLDLAST'  'Sold Last (late reporting)'
...

>> disp(struct2table(data))

    id      name      description
    ____  _____  _____
    1    'REGULAR'    'Normal Trade'
    2    'ACQ'       'Acquisition'
    3    'CASHM'     'Cash Only Market'
    4    'BUNCHED'   'Bunched Trade'
    5    'AVGPRI'    'Average Price Trade'
    6    'CASH'      'Cash Trade (same day clearing)'
    7    'DIST'      'Distribution'
    8    'NEXTDAY'   'Next Day Market'
    9    'BURSTBSKT' 'Burst Basket Execution'
   10    'BUNCHEDSOLD' 'Bunched Sold Trade'
   11    'ORDETAIL'  'Opening/Reopening Trade Detail'
   12    'INTERDAY'  'Intraday Trade Detail'
   13    'BSKTONCLOSE' 'Basket Index on Close'
   14    'RULE127'   'Rule - 127 Trade NYSE'
   15    'RULE155'   'Rule - 155 Trade AMEX'
   16    'SOLDLAST'  'Sold Last (late reporting)'
...
```

You can narrow-down the results by specifying **ID**, **Name** and/or **Description** filtering parameters. For example, to list only conditions whose **Description** contains ‘Option’:

```
>> struct2table(IQC('lookup', 'DataType','conditions', 'Description','option'))
ans =
7x3 table
    id      name      description
    ---  -
    39    'SPRD'    'Spread - Trade in Two Options in the Same Class
(a buy and a sell in the same class)'
    40    'STDL'    'Straddle - Trade in Two Options in the Same Class
(a buy and a sell in a put and a call)'
    43    'BWRT'    'Option Portion of a Buy/Write'
    44    'CMBO'    'Combo - Trade in Two Options in the Same Options
Class (a buy and a sell in the same class)'
    68    'STKOPT_TRADE'    'Stock-Option Trade'
    82    'OPTION_EX'    'Option Exercise'
    96    'OPT_ADDON'    'Short Option Add-On'
```

Naturally, it is quite possible that no security types exist that match the requested criteria. In such a case, the result will be empty (and cannot be displayed using Matlab’s struct2table() or struct2cell() functions):

```
>> data = IQC('lookup', 'DataType','conditions', 'Name','xyz')
data =
[]

>> struct2cell(data)
Undefined function 'struct2cell' for input arguments of type 'double'.
```

Note that the trade condition codes are typically reported by IQFeed as a string of one or more 2-digit hexadecimal values.¹⁴⁷ For example (see §4.1):

```
>> data = IQC('quotes', 'Symbol','GOOG')
data =
...
Most_Recent_Trade_Conditions: '3D87'
Trade_Conditions_Description: 'Intramaket Sweep; Odd lot trade'
```

In this example, the reported last trade had 2 trade conditions: hexadecimal 3D (=61, meaning 'Intramaket Sweep')¹⁴⁸ and hexadecimal 87 (=135, meaning 'Odd lot trade').

Here is a summary of the *IQC* parameters that affect trade conditions lookup:

Parameter	Data type	Default	Description
ID	integer or numeric array	[] (empty)	Limits the query to the specified ID(s).
Name	string	" (empty string)	Limits the query to trade conditions that contain the specified string in their name (case insensitive, <i>anywhere</i> within the name)
Description	string	" (empty string)	Limits the query to trade conditions that contain the specified string in their description (case insensitive, <i>anywhere</i> in the description)

¹⁴⁷ Trade condition codes 15 or lower are reported with a leading 0, e.g. 05 or 0E. The availability of the codes’ translation in the Trade_Conditions_Description field depends on **MsgParsingLevel**=2 (which is the default value; see §3.2).

¹⁴⁸ The missing “r” in “Intramarket” is a typo in IQFeed’s data

9 Connection, administration and other special commands

9.1 Connecting & disconnecting from IQFeed

When using *IQC*, there is no need to worry about connecting or disconnecting from IQFeed – *IQC* handles these activities automatically, without requiring user intervention. The user only needs to ensure that IQFeed is active and logged-in when the *IQC* command is invoked in Matlab.

IQC does not require any special configuration when connecting to IQFeed. It uses whatever setting was previously set in the DTN *IQConnect* client application. You might be prompted to enter a username/password, if *IQConnect* was not set up to automatically connect using saved login/password information:

In addition to entering the login credentials in the client window, you can also specify them programmatically. This could be useful when you have several IQFeed accounts and wish to switch between them programmatically, or if you use IQFeed's non-Windows client installer on MacOS (which prevents user-entry in the login window):

```
>> IQC('time', 'Username', '123456-1', 'Password', 'OpenSesame')
```

Note that the **Username** and **Password** parameters must be specified together, and that they are only meaningful in the first *IQC* command that starts the connection. If the **Username** and **Password** parameters are specified after a connection to IQFeed has been established, they will be ignored for the current connection, but stored in IQFeed's registry for subsequent connections (see §9.5).

If you enter an invalid set of **Username/Password**, an error message will be thrown. A different error will be thrown if *IQC* fails to connect to IQFeed within 10 seconds.

IQC can connect to a running IQFeed client, that was already started by another process on the current computer (e.g. charting app or another Matlab process that runs *IQC*), even without **Username** and **Password** in the initial *IQC* connection. *IQC* will bypass login, connecting directly to the client process.

IQC typically uses the latest API features supported by your installed IQFeed client (*IQConnect*). For example, if you use client version 6.0.1.1, *IQC* will use IQFeed communication protocol 6.0, and you will not have access to features of protocol 6.1.¹⁴⁹ If you install a newer IQFeed client, *IQC* will automatically detect this and use a newer protocol, as determined by the new client version. For debugging purposes, you can override **Protocol** to a version older than your installed client. For example, with a 6.0.1.1 client, you can set **Protocol** to 5.2 or 6.0, but not to 6.1 or newer:

```
>> IQC(..., 'Protocol', 6.1)
Warning: Your IQFeed client (version 6.0.1.1) does not support Protocol 6.1 -
using Protocol 6.0 instead
(Type "warning off IQC:UnsupportedProtocol" to suppress this warning.)
```

You will be able to retrieve information in Matlab as soon as *IQC* connects to the IQFeed client and [if necessary] the client finishes the login process and synchronizes with the IQFeed servers. This process typically takes a few short seconds.

The following parameters affect the initial connection to IQFeed:

Parameter	Data type	Default	Description
Username	string	(none)	Used to launch and login into <i>IQConnect</i>
Password	string	(none)	Used to launch and login into <i>IQConnect</i>
Protocol	number	installed client's version	IQFeed API protocol that should be used. You can set any protocol value supported by your <i>IQConnect</i> client version. For example, client 6.0.1.1 supports protocols 5.2 and 6.0 (this client's default), but not 6.1.

By default, *IQC* uses the following TCP ports to connect to IQFeed's client:

- 5009 – Level1Port – used for Level 1 queries/messages (quotes, fundamentals)
- 9100 – LookupPort – used for historical, news and lookup queries/messages
- 9200 – Level2Port – used for Level II (market-depth) queries/messages
- 9300 – AdminPort – used for administrative queries/messages (stats etc.)
- 9400 – DerivativePort – used for interval-bars queries/messages
- 60020 – LoginPort – used for login authentication

If any of these ports are already used by any other process at the time that IQFeed's client application (*IQConnect*) starts, it will not be able to communicate with *IQC* via these ports. *IQC* automatically tries to detect and report such cases, for example:

```
>> data = IQC('history', 'Symbol', 'IBM')
Error using IQC
Port #9100 is already used by process #12345 (SomeProcess.exe) and cannot be
used by IQFeed. Run IQC('registry') to set a different value for LookupPort.
```

You can solve such port conflicts using any of the following methods:

¹⁴⁹ For example, historic market summary data and scanning (§5.6)

1. Stop/uninstall the other process (`SomeProcess.exe` in this example)
2. Modify the other process's settings to use different ports, so that they will not conflict with IQFeed (this is rarely possible, but is preferable when possible)
3. Modify IQFeed's registry settings to use a different port, which is not used by any other process (see §9.5). If you choose this option, you will need to fix the port again whenever you re-install the IQFeed client (for example, whenever you update the client to a newer version), because the client's installation process resets the port values back to their default values.



After you resolve the problem using one of these alternatives, restart IQFeed's *IQConnect* client and then retry the *IQC* command. An *IQConnect* restart is necessary since *IQConnect* reads the registry values and sets the ports only when it starts.

You can close a live connection to IQFeed using the 'disconnect' action:

```
>> IQC('disconnect')
```

This command disconnects *IQC* from the IQFeed client (*IQConnect*). If *IQC* was the only application that was connected to the client, then the client will silently exit after several seconds, unless a new IQFeed connection is established during this time. If *IQConnect* is not active, the 'disconnect' action is silently ignored (no error).

There is no need for a corresponding 'connect' action, because IQFeed connection is automatically (re-)established whenever this is required by a new *IQC* command.

IQC and *IQConnect* automatically try to recover from connection losses during normal operation. You may see in the Matlab console one or more *IQConnect* error messages such as the following, which indicate such a connection loss:

```
20180410 20:03:06.371 Level1 server disconnected!
OR:
20180410 20:03:57.934 Unable to connect to L2IP server.  Error Code: 10051
Error Msg: A socket operation was attempted to an unreachable network.
OR:
20180410 20:03:57.934 Unable to connect to L2IP server.  Error Code: 10065
Error Msg: A socket operation was attempted to an unreachable host.
OR:
20180410 20:03:57.934 Unable to connect to L2IP server.  Error Code: 10053
Error Msg: An established connection was aborted by the software in your host
machine.
OR:
20180410 20:03:57.934 Unable to connect to L2IP server.  Error Code: 10060
Error Msg: A connection attempt failed because the connected party did not
properly respond after a period of time
```

You can safely ignore such messages in most cases, since *IQConnect* will automatically re-establish connection with IQFeed's servers as soon as they become accessible again, and show an appropriate informational message in Matlab's console:

```
20180410 20:04:16.497 Level1 server is connected
```

You can actively disconnect and then connect to IQFeed using the 'reconnect' action:

```
>> IQC('reconnect')
```



Note that after reconnecting to IQFeed, you will need to request any and all streaming data again (see §6), since IQFeed resets data streaming after a client disconnect.

9.2 Server time

You can request the latest IQFeed server time using the 'time' action:

```
>> data = IQC('time')
data =
    latestEventDenum: 737114.660205451
    latestEventTimeStamp: '20180223 15:50:41'
    latestServerDenum: 737114.368518519
    latestServerTimeStamp: '20180223 08:50:40'
```

The returned data struct includes the following data fields:

- `latestEventDenum` – a Matlab numeric datenum value that corresponds to the **local** time in which the very latest message has arrived from IQFeed.
- `latestEventTimeStamp` – a human-readable format of `latestEventDenum`
- `latestServerDenum` – a Matlab numeric datenum value that corresponds to the latest **server** time that was received from IQFeed.
- `latestServerTimeStamp` – a human-readable format of `latestServerDenum`

Note that the server time may be off by up to a full second from the current time, depending on when the last timestamp message was received from IQFeed. IQFeed sends server time messages once every second, so `latestServerDenum` lags by 0.5 secs behind the current time on average.

Similarly, `latestEventDenum` reports the time at which the last message was received from IQFeed. This message could be a timestamp message, or any other data message. For this reason, the lag here is typically much lower than the lag of `latestServerDenum`.

The 'time' action has no settable properties.

9.3 Client stats

You can retrieve the updated IQFeed connection traffic stats using the 'stats' action:

```
>> data = IQC('stats')
data =
    ServerIP: '66.112.148.226'
    ServerPort: 60002
    MaxSymbols: 1300
    NumOfStreamingSymbols: 0
    NumOfClientsConnected: 3
    SecondsSinceLastUpdate: 1
    NumOfReconnections: 0
    NumOfAttemptedReconnections: 0
    StartTime: 'Mar 07 11:03AM'
    MarketTime: 'Mar 07 04:34AM'
    ConnectionStatus: 'Connected'
    IQFeedVersion: '5.2.7.0'
    LoginID: '123456-1'
    TotalKBsRecv: 42.98
    KBsRecvLastSecond: 0.02
    AvgKBsPerSecRecv: 0.02
    TotalKBsSent: 361.62
    KBsSentLastSecond: 0.22
    AvgKBsPerSecSent: 0.19
    Exchanges: {1x16 cell}
    ServerVersion: '6.0.0.5'
    ServiceType: 'real_time'
```

The returned data struct includes the following data fields:¹⁵⁰

- **ServerIP** – IP address of the least loaded IQFeed Quote Server
- **ServerPort** – Port number for least loaded IQFeed Quote Server
- **MaxSymbols** – The maximum # of symbols that can be streamed simultaneously
- **NumOfStreamingSymbols** – # of symbols that are currently being streamed
- **NumOfClientsConnected** – # of clients that are currently connected
- **SecondsSinceLastUpdate** – # of seconds since last update from the Quote Server
- **NumOfReconnections** – # of times that IQFeed successfully reconnected
- **NumOfAttemptedReconnections** – # of times IQFeed failed to reconnect
- **StartTime** – Time of latest connection/reconnection to IQFeed (local timezone)
- **MarketTime** – Current time of the market (market's time-zone)
- **ConnectionStatus** – Represents whether IQFeed is connected or not
- **IQFeedVersion** – Represents the version of IQFeed that is running
- **LoginID** – The Login ID that is currently logged into IQFeed
- **TotalKBsRecv** – Total # of Kilobytes received by IQFeed from *IQC* (i.e., *IQC* commands/requests to IQFeed). Found in the “Internet Bandwidth” section of the IQConnection Manager. Formula: total bytes received / 1024
- **KBsRecvLastSecond** – Found in the “Internet Bandwidth” section of the IQConnection Manager. Formula: bytes received in the past second / 1024
- **AvgKBsPerSecRecv** – Found in the “Internet Bandwidth” section of the IQConnection Manager. Formula: total KB's received / total seconds

¹⁵⁰ <http://iqfeed.net/dev/api/docs/AdminSystemMessages.cfm>

- **TotalKBsSent** – Total # of Kilobytes sent from IQFeed to *IQC* (i.e., IQFeed messages to *IQC*). Found in the “Local Bandwidth” section of the IQConnection Manager. Formula: total bytes sent / 1024
- **KBsSentLastSecond** – Found in the “Local Bandwidth” section of the IQConnection Manager. Formula: bytes sent in the past second / 1024
- **AvgKBsPerSecSent** – Found in the “Local Bandwidth” section of the IQConnection Manager. Formula: total KB's sent / total seconds.
- **Exchanges** – A cell-array list of exchanges and news subscriptions for which this IQFeed account is subscribed. Delayed data has 'DL-' prefix. For example: {'DL-FTSE', 'EUREXNDX', 'DL-NYMEX', 'CBOENDX', 'BLOOMBERG', 'FOREX_PREMIUM', 'RT_TRADER', 'DTNNEWS', 'BENZINGA', 'INDEX'}
- **ServerVersion** – The current version of IQFeed that the server supports. This is always the same or higher than your locally-installed *IQFeedVersion*.
- **ServiceType** – Type of data provided for this account (delayed or real-time)

The 'stats' action has a single settable property: **AddPortStats** (default=0). If you set this property to 1 or true, then additional stats will be returned, with extra information on the connected data channels/ports (see the highlighted fields below):

```
>> data = IQC('stats', 'AddPortStats', 1)
data =
    ServerIP: '66.112.148.224'
    ServerPort: 60005
    MaxSymbols: 1300
    NumOfStreamingSymbols: 0
    NumOfClientsConnected: 4
    SecondsSinceLastUpdate: 0
    NumOfReconnections: 0
    NumOfAttemptedReconnections: 0
    StartTime: 'Apr 01 8:21PM'
    MarketTime: 'Apr 01 02:12PM'
    ConnectionStatus: 'Connected'
    IQFeedVersion: '5.2.7.0'
    LoginID: '464720-1'
    TotalKBsRecv: 69.44
    KBsRecvLastSecond: 0.04
    AvgKBsPerSecRecv: 0.02
    TotalKBsSent: 1470.32
    KBsSentLastSecond: 0.47
    AvgKBsPerSecSent: 0.48
    Exchanges: {1x16 cell}
    ServerVersion: '6.0.0.5'
    ServiceType: 'real_time'
    Level2: [1x1 struct]
    Level2SymbolsWatched: 2
    Lookup: [1x1 struct]
    RegionalSymbolsWatched: 2
    Admin: [1x1 struct]
    Level1: [1x1 struct]
    Level1SymbolsWatched: 0

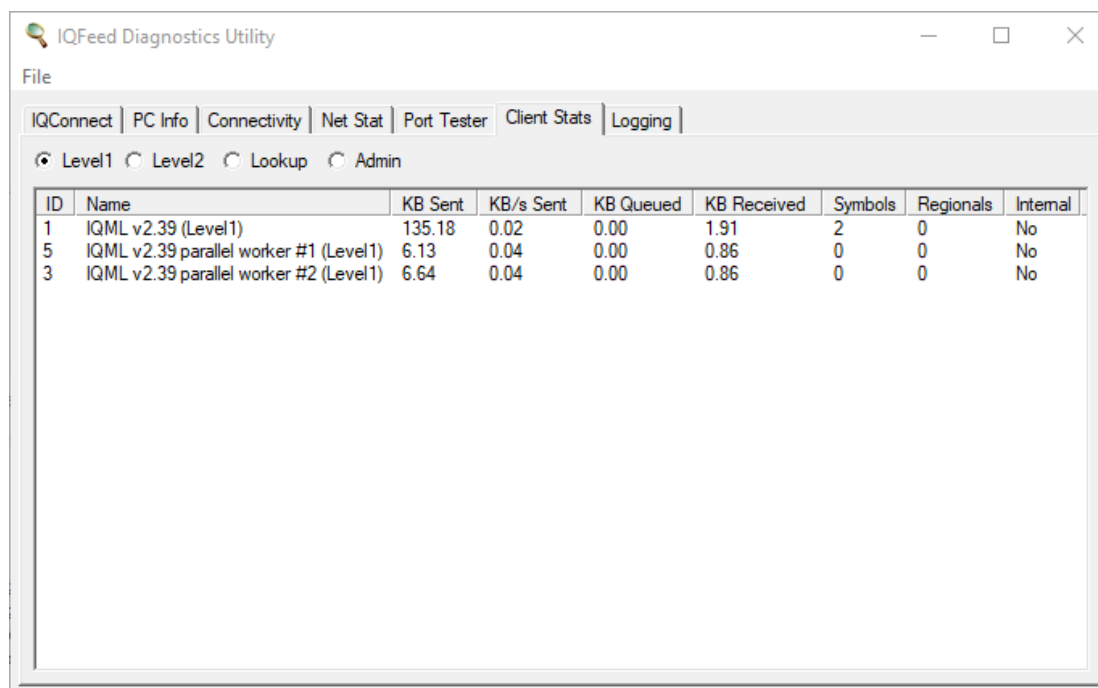
>> data.Level1
ans =
    ConnectTime: '20180401 202111'
    KBsReceived: 0.74
    KBsSent: 70.58
    KBsQueued: 0
```

Note that it might take a few seconds for the additional port stats to arrive after the initial command. If you don't see the expected results immediately, simply re-query them after 1-2 secs.

The returned data structs include the following data fields, separately for each data channel (port):¹⁵¹

- `ConnectTime` – Timestamp when this channel to IQFeed was first opened
- `KBsReceived` – Total # of Kilobytes sent from *IQC* to IQFeed via this channel (requests for data queries etc.).
- `KBsSent` – Total # of Kilobytes sent from IQFeed to *IQC* via this channel. This data transfer is typically much larger than `KBsReceived` – this is normal and does not indicate a problem.
- `KBsQueued` – Total # of Kilobytes waiting in *IQConnect* to be sent to *IQC* via this channel for processing. This value should typically be 0 (zero) – a consistent non-zero value indicates that the Matlab program is unable to keep up with the inflow of data from IQFeed, perhaps due to a high load on the computer, or some heavy processing of the incoming data. If the value increases over time, Matlab and your computer will eventually freeze and become non-responsive, requiring a hard reset. See §3.6 for ways to speed-up the processing time, in order to get `KBsQueued` back to 0.

Corresponding information can also be seen in *IQConnect*'s Diagnostics Utility, which is installed as part of IQFeed's client installation:



ID	Name	KB Sent	KB/s Sent	KB Queued	KB Received	Symbols	Regionals	Internal
1	IQML v2.39 (Level1)	135.18	0.02	0.00	1.91	2	0	No
5	IQML v2.39 parallel worker #1 (Level1)	6.13	0.04	0.00	0.86	0	0	No
3	IQML v2.39 parallel worker #2 (Level1)	6.64	0.04	0.00	0.86	0	0	No

¹⁵¹ See §9.1 for a description of IQFeed's data channels (ports)

9.4 Sending a custom command to IQFeed

You can send any custom command to IQFeed's API, using the 'command' action. For example, to send the 'S,TIMESTAMP SOFF' command,¹⁵² which stops IQFeed from sending server timestamp messages every second:

```
>> IQC('command', 'String', 'S,TIMESTAMP SOFF')
```

IQFeed expects that users send commands to its API via specific channels ("ports"). Each command is typically accepted only by the port for which it is defined. For example, the 'S,TIMESTAMP SOFF' command is defined for the Level1 port,¹⁵³ whereas the 'S,CLIENTSTATS OFF' command (which stops the IQFeed server from streaming client stats messages) is defined for the Admin port.¹⁵⁴ When you use *IQC*'s standard actions you do not need to worry about which port handles which command – this is automatically handled by *IQC*. But when you send a custom command to IQFeed, you need to specify the port, if it is different from the default ('Level1'). In this specific example:

```
>> IQC('command', 'String', 'S,CLIENTSTATS OFF', 'PortName', 'Admin')
```

IQFeed is very picky about the spelling of the commands, including spaces and casing. If the spelling is not exactly right, the command will be rejected by IQFeed, possibly even without an error message. Unfortunately, IQFeed is not always consistent in the format of the various commands. For example, the 'S,TIMESTAMP SOFF' command has no space, whereas the 'S,CLIENTSTATS OFF' command does include a space; also, both of these commands are all-uppercase, whereas the 'S,SET AUTOCONNECT,On' Admin command spells On/Off with lowercase letters (and uses a comma instead of a second space).

In some cases, the command that is sent to IQFeed may result in data messages that will be sent back from IQFeed, which should be received and processed. To do this, you can set the **ProcessFunc** property to a custom callback function that will handle these messages (see §10).

The following properties can be specified in IQC with the 'command' action:

Parameter	Data type	Default	Description
String	string or cell-array of strings	(none)	The IQFeed command string(s).
PortName	string	'Level1'	The IQFeed port that will process the command(s). Must be one of the following: <ul style="list-style-type: none"> • 'Level1' (default) • 'Level2' • 'Lookup' • 'Admin'
ProcessFunc	function handle	[]	Custom user callback function to process incoming IQFeed data messages (see §10).

¹⁵² <http://iqfeed.net/dev/api/docs/Level1viaTCPIP.cfm>

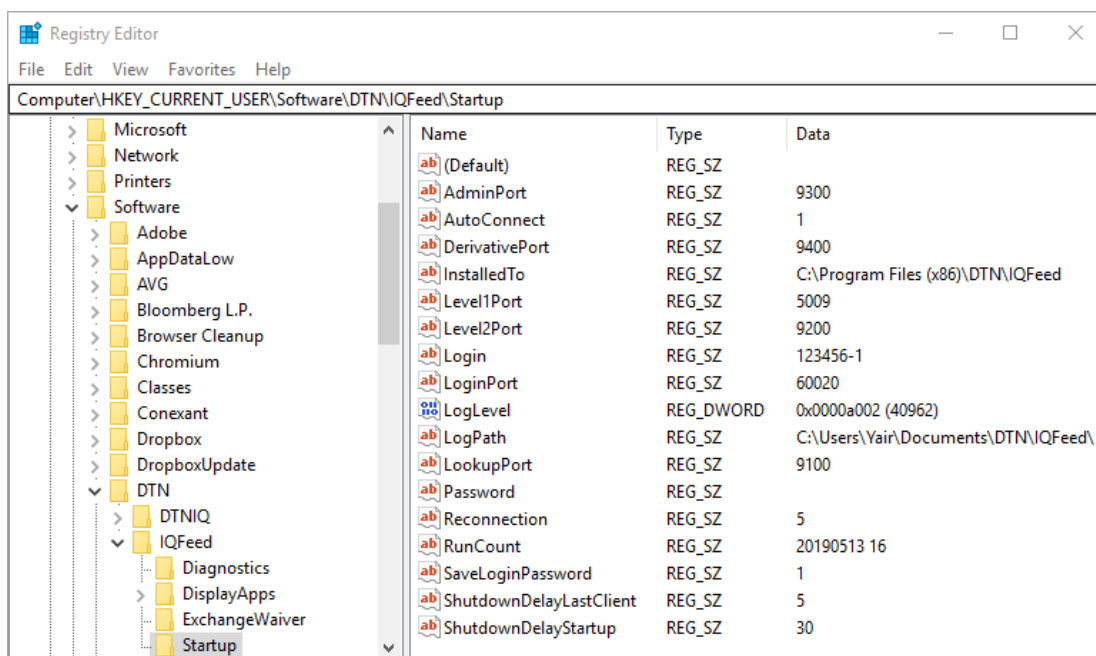
¹⁵³ <http://iqfeed.net/dev/api/docs/Level1viaTCPIP.cfm>

¹⁵⁴ <http://iqfeed.net/dev/api/docs/AdminviaTCPIP.cfm>

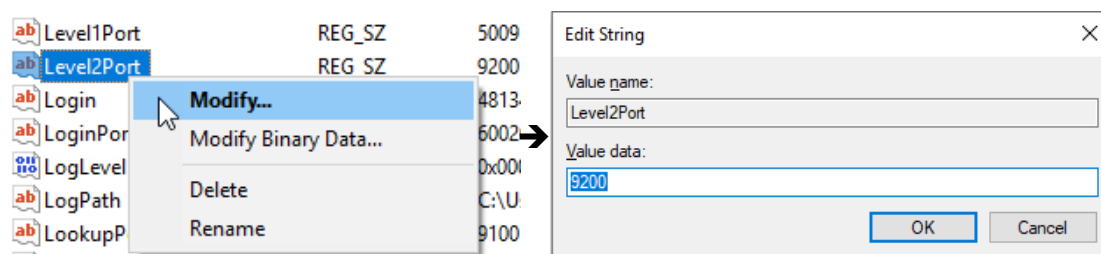
9.5 Modifying IQFeed's registry settings

IQFeed stores its settings as keys in the HKEY_CURRENT_USER\Software\DTN\ path of the Windows registry. Most of the important setting keys that a user might want to modify are located within HKEY_CURRENT_USER\Software\DTN\IQFeed\Startup\ . These registry keys can be reviewed and modified using Window's built-in Registry Editor (*regedit.exe*) utility. As a convenient method to open the Registry Editor at the correct path, use the *IQC* 'registry' action, which has no settable parameters: ¹⁵⁵

```
>> IQC('registry')
```



IQFeed's key names are generally self-explanatory. In most cases you should leave the settings unchanged. Some circumstances where you might want to modify certain keys are discussed in §9.1 and §12.2. Key values can be modified by right-clicking the key name and selecting "Modify...":



Be **EXTREMELY** careful when editing the Windows registry: if you make a mistake it could disable not only a specific program, but possibly even Windows itself, rendering the computer useless. Never delete or rename keys, only modify their value. If you are unsure which registry key to modify and how, ask DTN's technical support.

¹⁵⁵ Note: running Windows Registry Editor requires local computer Administrator privileges (or elevation).
In Mac/Linux, IQC('registry') will naturally work only when Matlab runs under Parallels/Wine, not in native mode.

10 Attaching user callbacks to IQFeed messages

10.1 Processing IQFeed messages in IQC

IQFeed uses an asynchronous event-based mechanism for sending information to clients. This means that we do not simply send a request to IQFeed and wait for the answer. Instead, we send a request, and when IQFeed is ready it will send us one or more (or zero) messages in response. Each of these messages evoke an event that carry data (the message content and the originating IQFeed channel/port-name). By analyzing the event data we (hopefully) receive the answer that we were waiting for.

Matlab has built-in support for asynchronous events, called *callbacks* in Matlab jargon.¹⁵⁶ Whereas Matlab callbacks are normally used in conjunction with Graphical User Interfaces (GUI), they are also used with *IQC*, which automatically converts all the IQFeed events into a Matlab callback invocation.

The callback that processes incoming IQFeed messages is constantly being “fired” (i.e., invoked) by asynchronous messages from IQFeed, ranging from client stats and time messages (once per second, for each of IQFeed’s 3 channels/ports), to system messages (e.g. connection losses and reconnections), to error messages and responses to market queries. Some of the events are triggered by user actions (market or portfolio queries, for example), while others are triggered by IQFeed (e.g., client stats once a second).

In addition to the regular *IQC* callback that processes all incoming IQFeed message events, you can assign your own custom Matlab function that will be triggered whenever a certain IQFeed message arrives. In all cases, the parameter controlling this callback in *IQC* is called **ProcessFunc**.

There are two types of callbacks that you can use in *IQC*:

- *Generic callback* – this is a catch-all callback function that is triggered upon any IQFeed message event. Within this callback, you would need to write some code to distinguish between the different event types in order to process the events’ data. A skeleton for this is given below.
- *Specific callback* – this is a callback function that is only triggered when the specific event type is received from IQFeed. Since the event type is known, you can process its event data more easily than in the generic callback case. You can specify a different specific callback for each of the event types that you wish to process, as well as a default callback that will be used for any other event that was not assigned a specific callback.

When you specify any callback function to *IQC*, the command/action does not need to be related to the callback. For example:

```
data = IQC('time', 'ProcessFunc', @IQC_Callback);
```

where `IQC_Callback()` is a Matlab function created by you that accepts two input arguments, which are automatically populated in run-time:

¹⁵⁶ http://www.mathworks.com/help/matlab/creating_guis/writing-code-for-callbacks.html

- `iqObject` – this is currently an empty array. Future versions of *IQC* may place an actual object in this argument.
- `eventData` – a Matlab struct that contains the event’s data in separate fields. This struct includes the following fields:
 - `Timestamp` – local time in Matlab numeric datenum format.
 - `MessagePort` – the name of the IQFeed port that sent the message: 'Level1', 'Level2', 'Lookup' or 'Admin'.
 - `MessageType` – the event type, which corresponds to the custom fields that can be set in the **ProcessFunc** parameter for *specific callbacks*.
 - `MessageHeader` – the first part of the message text string, that identified the message type. This is typically used to set the `MessageType` field.
 - `MessageString` – the message text string as received from IQFeed.
 - `MessageParts` – processed parts of `MessageString`, as a cell-array.

An example of defining a Matlab callback function is:

```
function IQC_Callback(iqObject, eventData)
    % do callback processing here using the info in eventData
end
```

You can pass external data to your callback functions using the callback cell-array format.¹⁵⁷ For example, to pass two extra data values:

```
callbackDefinition = {@IQC_Callback, 123, 'abc'};
IQC('time', 'ProcessFunc', callbackDefinition);

function IQC_Callback(iqObject, eventData, extra1, extra2)
    % do callback processing here using the info in eventData, extra1, extra2
end
```

Here are examples of the `eventData` for two different IQFeed messages – a timestamp message (sent from IQFeed once every second on the Level1 and Level2 ports), and a connection stats message (sent from IQFeed once a second on the Admin port):

```
Timestamp: 737128.675475417
MessagePort: 'Level1'
MessageType: 'Time'
MessageHeader: 'T'
MessageString: 'T,20180309 09:12:39'
MessageParts: {'T' '20180309 09:12:39'}

Timestamp: 737128.675479248
MessagePort: 'Admin'
MessageType: 'System'
MessageHeader: 'S'
MessageString: 'S,STATS,66.112.148.225,60002,1300,0,4,0,0,0,Mar 09
3:10PM,Mar 09 09:12AM,Connected,5.2.7.0,464720-
1,86.43,0.04,0.02,759.37,0.20,0.20'
MessageParts: {1x20 cell}
```

All IQFeed messages typically begin with a single character followed by ‘,’, which we call the `MessageHeader`, that identifies the `MessageType`. In the examples above, the `MessageHeader` of the first message is 'T' (indicating a Time message), and in the second message it is 'S' (indicating a System message).¹⁵⁸

¹⁵⁷ http://www.mathworks.com/help/matlab/creating_guis/writing-code-for-callbacks.html#brqow8p

¹⁵⁸ An exception to this rule may happen if you send custom commands to IQFeed using the mechanism in §7.4. In such case, it is possible that `MessageHeader` will not be a recognized or even a single character. It will have a `MessageType` of 'Other'.

All the callbacks examples so far have set a *generic* callback that is used for all incoming IQFeed messages. As noted above, you can also set *specific* callbacks for specific messages. For example:

```
% Alternative #1: using the struct() function:
>> callbacks = struct('Time','disp TIME!', ...
    'System',@(h,e)disp(e.MessageString));

% Alternative #2: using direct field assignments:
>> callbacks.Time = 'disp TIME!';
>> callbacks.System = @(h,e)disp(e.MessageString);

>> IQC('time','processFunc',callbacks);
TIME!
TIME!
S,STATS,66.112.156.228,60002,1300,0,4,0,0,1,Mar 11 12:36PM,Mar 11
07:14AM,Connected,5.2.7.0,464720-1,51.51,0.04,0.02,516.30,0.23,0.23
TIME!
TIME!
S,STATS,66.112.156.228,60002,1300,0,4,0,0,1,Mar 11 12:36PM,Mar 11
07:14AM,Connected,5.2.7.0,464720-1,51.54,0.04,0.02,516.48,0.23,0.23
TIME!
```

In this example, we have set two separate custom callbacks for two different IQFeed messages: the periodic timestamp messages and the periodic system update messages.

In addition to specific callbacks for specific message types, you can also set a “Default” callback that will be invoked for each incoming IQFeed message that does not have a specific callback assigned to it.

The following message types can be trapped, corresponding to the eventData’s MessageType field (e.MessageType):

MessageType	Message Header	Description	See section
Fundamental	F	Fundamental asset data	§4.2
Quote_summary	P	Quote summary message	§4.1
Quote_update	Q	Quote update (tick) message	§6.1
Market_depth	Z	Level2 market-depth update message	§4.4 , §6.4
Market_maker	M	Market maker information	§4.4 , §6.4
History	H	Historical data (one msg per bar/tick)	§5
Regional	R	Regional update message	§6.2
News	N	News data (one message per item)	§7
End_of_data	!ENDMSG!	Indicates end of the data with multiple data items (e.g., history or news)	§5 , §7
Lookup	s	Lookup information message	§8.1
Chain	:	Options/Futures chain	§8.2
Time	T	Timestamp message (once a second)	§9.2
System	S	System message (stats, once a sec)	§9.3
Symbol_not_found_error	n	Indicates a symbol-not-found error	§3.4
General_error	E	All other IQFeed-generated errors	
Other		All other IQFeed messages	
Default		Any IQFeed message that does not have a specific callback assigned to it	

You can set individual callbacks to any of these `MessageType` values, by using the `MessageType` value as a field-name in the **ProcessFunc** parameter. For example, to process quote-update (tick) messages in a dedicated callback function:

```
>> callbacks.Quote_update = @IQC_Quote_Update_Callback;
>> IQC('time', 'ProcessFunc', callbacks);
```

Here is a more elaborate example, where we set different callbacks for different message types, along with a default callback for all other message types:

```
% Alternative #1: using the struct() function:
>> callbacks = struct('Time','disp TIME!', ...
    'System',[], ... % ignore System messages
    'Quote_update',@IQC_Quote_Update_Callback, ...
    'Default',@IQC_Default_Callback);

% Alternative #2: using direct field assignments:
>> callbacks.Time = 'disp TIME!';
>> callbacks.System = []; % ignore System messages
>> callbacks.Quote_update = @IQC_Quote_Update_Callback;
>> callbacks.Default = @IQC_Default_Callback);

>> IQC('time', 'processFunc', callbacks);
```

When you specify any callback function to *IQC*, you only need to set it once, in any *IQC* command. Unlike most *IQC* parameters, which are not remembered across *IQC* commands and need to be re-specified, callbacks do not need to be re-specified. They are remembered from the moment they are first set, until such time as Matlab exits or the callback parameter is changed.

Note that it is not an error to re-specify the callbacks in each *IQC* command, it is simply useless and makes the code less readable.

If an error occurs during the evaluation of your specified callback function, an error message will be displayed in the Matlab console. In blocking mode (`data=IQC(...)`) a Matlab exception will then be thrown, which can be trapped in the calling code using a try-catch block (see §3.4 item 1); in non-blocking mode no exception will be thrown:

```
>> IQC('quotes', 'symbol', 'IBM', 'processFunc', struct('Quote_Summary',@myFunc))
20200330 11:02:55.510 error in user-defined callback (myFunc) for IQFeed
Quote_Summary message: Invalid argument in myFunc line 123
```

To reset all callbacks (i.e., remove any callback invocation), simply set the **ProcessFunc** parameter value to `[]` (empty square brackets):

```
IQC('time', 'ProcessFunc', []);
```

You can also set individual message callbacks to an empty value, in order to ignore just these specific messages but not the other messages:

```
>> callbacks.Time = 'disp TIME!';
>> callbacks.System = []; % ignore System messages
>> callbacks.Default = @IQC_Default_Callback);

>> IQC('time', 'ProcessFunc', callbacks);
```

Matlab callbacks are invoked even if you issue a custom *IQFeed* command (see §9.4). This can be very useful: you can send a request to *IQFeed*, and then process the results in a Matlab callback function. However, note that in such a case, it is possible that the returned message will contain a `MessageHeader` that will not be a recognized. Such messages will be assigned a `MessageType` of 'Other'.

10.2 Run-time performance implications

It is very important to ensure that any callback function that you assign in *IQC* completes in the fastest possible time. This is important for programming in general, but it is especially important for *IQC* callbacks, which are invoked (executed) every time that a new message arrives from IQFeed, numerous times each second.

As explained in §3.6, *IQC*'s standard callback processing has an overhead of 1-2 milliseconds per IQFeed message. This means that without any user-specified callbacks, and without any other Matlab or other code running, *IQC* can typically process up to 500-1000 IQFeed messages per second.

When you add your own user-defined callbacks, their processing time is added to *IQC*'s. For example, if your callback takes an average of just 3 msecs to process (which is quite fast), the total average message processing time will be 4-5 msecs. This will lower *IQC*'s effective processing rate from 500-1000 messages/sec to just 200-250 messages/sec. The more callbacks and alerts that you define, and the longer each of them takes to process, the lower *IQC*'s message processing rate will be.

The following specific tips may assist you to reduce the callback performance impact:

1. Ensure that you have enough physical memory to avoid memory swapping to disk. This is probably the single most important tip to improve performance
2. Avoid setting user callbacks and alerts, or at least disable them when not needed.
3. Avoid setting a Default callback or a general **ProcessFunc**, but rather specific callbacks only for the messages that you need (e.g. for News or Regional).
4. Limit the streaming data to just those events and symbols that are of interest to you. For example, if you are only interested in the GOOG symbol, and set a Quote_update callback, this callback will also be processed for streaming quotes for other symbols, so it's better to stop streaming those other symbols.
5. Minimize disk access: disk I/O is much slower than memory access. Save data to memory and flush it to disk at the end of the trading day, or once in a while (e.g. every 5 minutes), but **not** in each callback.
6. If you need to access disk files, use SSD (solid-state disk) rather than a spinning hard-disk.
7. If you need to load data from disk, do it once and preserve the data in memory using Matlab `persistent` or `global` variables, to be reused in callback calls.
8. Instead of re-computing values that are based on static data in each callback call, compute once and cache results in Matlab `persistent` or `global` variables.
9. Use Matlab's built-in Profiler tool¹⁵⁹ to check your callback code for run-time performance hotspots that can be optimized to run faster.
10. Read the textbook "*Accelerating MATLAB Performance*",¹⁶⁰ authored by *IQC*'s creator (see §15.2), for numerous tips on improving Matlab run-time.

¹⁵⁹ https://mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html

10.3 Usage example – using callbacks to parse options/futures chains

In this example, we request IQFeed to send the list of symbols in an options/futures chain, then parse the incoming results to retrieve the symbols in the chain (see §8.2).

We first send the relevant command to IQFeed using *IQC*'s custom command functionality (§9.4), specifying a custom callback function for the 'Chain' MessageType:¹⁶¹

```
% Equity options chain for GOOG:
processFunc.Chain = @IQC_Chain_Callback;
>> IQC('command', 'String', 'CEO,GOOG,p,,1', 'PortName','lookup', ...
      'debug',1, 'ProcessFunc',processFunc)

=> 20180405 13:13:00.063 (lookup) CEO,GOOG,p,,1
<= 20180405 13:13:00.574 (lookup) :,GOOG1806P1000,GOOG1806P1002.5,GOOG1806P1005,GOOG1806P1007.5,GOOG1806P1010,GOOG1806P1012.5,GOOG1806P1015,GOOG1806P1017.5,GOOG1806P1020,GOOG1806P1022.5,GOOG1806P1025,GOOG1806P1027.5,GOOG1806P1030,GOOG1806P1032.5,GOOG1806P1035,GOOG1806P1037.5,GOOG1806P1040,GOOG1806P1042.5,GOOG1806P1045,GOOG1806P1047.5,GOOG1806P1050,...,
<= 20180405 13:13:00.578 (lookup) !ENDMSG!

% Future options chain for C:
>> IQC('command', 'String', 'CFO,C,p,,9,1', 'PortName','lookup', ...
      'debug',1, 'ProcessFunc',processFunc)

=> 20180405 13:31:48.677 (lookup) CFO,C,p,,9,1
<= 20180405 13:31:49.149 (lookup) :,CH19P2000,CH19P2100,CH19P2200,CH19P2300,CH19P2400,CH19P2500,CH19P2600,CH19P2700,CH19P2800,CH19P2900,CH19P3000,CH19P3100,CH19P3200,CH19P3300,CH19P3400,CH19P3500,CH19P3600,CH19P3700,CH19P3800,CH19P3900,CH19P4000,CH19P4100,CH19P4200,CH19P4300,CH19P4400,CH19P4500,CH19P4600,CH19P4700,CH19P4800,CH19P4900,CH19P5000,CH19P5100,CH19P5200,CH19P5300,CH19P5400,CH19P5500,CH19P5600,CH19P5700,CH19P5800,CH19P5900,CH19P6000,CH19P6100,CH19P6200,CH19P6300,CH19P6400
<= 20180405 13:31:49.158 (lookup) !ENDMSG!
```

The custom callback function may look something like this:

```
function IQC_Chain_Callback(iqObject,eventData)
    symbols = eventData.MessageParts(2:end); %discard the ':' message header
    % now do something useful with the reported symbols...
end
```

¹⁶¹ Note that we have set **Debug**=1 in this example purely to illustrate the incoming IQFeed message format; it would not be used in a typical run-time program.

10.4 Usage example – using callbacks for realtime quotes GUI updates

In this example, we wish to update a real-time ticker window with the latest streaming quotes data. The idea is to create a minimalistic window and update its title with the symbol name and latest trade price, whenever a new tick arrives.

The code relies on the format of IQFeed's Quote_update (Q) message, which by default is a 16-element cell array: {Symbol, Most_Recent_Trade, Most_Recent_Trade_Size, Most_Recent_Trade_Time, Most_Recent_Trade_Market_Center, Total_Volume, Bid, Bid_Size, Ask, Ask_Size, Open, High, Low, Close, Message_Contents, Most_Recent_Trade_Conditions}:

```
>> processFunc = struct('Quote_Update', @Quote_Update_Callback);
>> IQC('quotes', 'symbol', '@VX#', 'numofevents', 100, ...
      'ProcessFunc', processFunc, 'debug', 1)

=> 20180411 12:03:40.131 (Level1) w@VX#
<= 20180411 12:03:40.391 (Level1) F,@VX#,20.61,,28.05,12.85,,,,,,,,,,,,,,,,,CBOE ...
<= 20180411 12:03:40.409 (Level1) P,@VX#,20.61,,04:52:29.711000,32,5668,20.60,50,
20.65,87,20.20,20.70,20.15,20.18,Cbasohlcv,4D
<= 20180411 12:03:44.887 (Level1) Q,@VX#,20.61,,04:52:29.711000,32,5668,20.60,50,
20.65,86,20.20,20.70,20.15,20.18,a,4D
```

In our case, we are only interested in the 1st (Symbol) and 2nd (Most_Recent_Trade) elements of the 'Q' update messages:

```
eventData =
    Timestamp: 737161.502602859
    MessagePort: 'Level1'
    MessageType: 'Quote_Buffer'
    MessageHeader: 'Q'
    MessageString: 'Q,@VX#,20.61,,04:52:29.711000,32,5668,20.60,50,20.65,86,
20.20,20.70,20.15,20.18,a,4D'
    MessageParts: {'@VX#' 20.61 [] '04:52:29.711000' 32 5668 20.6 50
20.65 86 20.2 20.7 20.15 20.18 'a' '4D'}
```

The corresponding callback function will be:

```
function Quote_Update_Callback(iqObject, eventData)
% Symbol is 1st data element of IQFeed 'Q' messages
symbol = eventData.MessageParts{1};

% Last trade price is 2nd data element of the IQFeed 'Q' messages
latestTrade = eventData.MessageParts{2};

% Get the handle for this symbol's ticker window
hFig = findall(0, 'Tag', symbol, '-depth', 1);
if isempty(hFig)
    % Ticker window not found, so create it
    hFig = figure('Tag', symbol, 'Position', [300,300,250,1], ...
        'Resize', 'off', 'NumberTitle', 'off', ...
        'Menu', 'none', 'Toolbar', 'none',);
end

% Update the ticker window's title
hFig.Name = sprintf('%s: %.2f', symbol, latestTrade);
end
```

And the resulting ticker window will look like this:



As noted in §6.1, tick events may be sent at a very high rate from the IQFeed server. So instead of updating the GUI with each tick, you may want to use a periodic Matlab timer having a Period of 0.5 [secs], that will invoke a timer callback, which will call `IQC(...,NumOfEvents',-1)` to fetch the latest data and update the GUI.

10.5 Usage example – using callbacks for realtime order-book GUI updates

In this example, we wish to update a real-time GUI display of the order-book (at least the top few rows of the book).



Note: Market Depth (Level 2) data is only available in the Professional IQC license.

As noted in §6.4, market-depth events may be sent at a very high rate from the IQFeed server, and so it is not feasible or useful to update the Matlab GUI for each update. Instead, we update the GUI with the latest data at a steady rate of 2 Hz (twice a second). This can be achieved in two different ways: one alternative is to set-up a periodic timer that will run our GUI-update callback every 0.5 secs, which will call `IQC(...,'NumOfEvents',-1)` to fetch the latest data and update the GUI.

Another alternative, shown here below (also downloadable¹⁶²), is to attach a user callback function to Level 2 market-depth messages, updating an internal data struct, but only updating the GUI if 0.5 secs or more have passed since the last GUI update:

```
% IQC_MktDepth - sample Market-Depth usage function
function IQC_MktDepth(symbol)

    % Initialize data
    numRows = 10;
    depthData = cell(numRows,8);
    lastUpdateTime = -1;
    GUI_refresh_period = 0.5 * 1/24/60/60; % =0.5 secs

    % Prepare the GUI
    hFig = figure('Name','IQC market-depth example', ...
        'NumberTitle','off','CloseReq',@figClosedCallback,...
        'Menubar','none','Toolbar','none', ...
        'Resize','off','Pos',[100,200,660,260]);
    color = get(hFig,'Color');
    headers = {'Ask valid','Ask time','Ask size','Ask price', ...
        'Bid price','Bid size','Bid time','Bid valid'};
    formats = {'logical','char','numeric','long', ...
        'long','numeric','char','logical'};
    hTable = uitable('Parent',hFig, 'Pos',[10,40,635,203], ...
        'Data',depthData, 'ColumnName',headers, ...
        'ColumnFormat',formats, ...
        'ColumnWidth',{60,100,80,80,80,80,100,60});
    hButton = uicontrol('Parent',hFig, 'Pos',[50,10,60,20], ...
        'String','Start', 'Callback',@buttonCallback);
    hLabel1 = uicontrol('Parent',hFig, 'Pos',[120,10,100,17], ...
        'Style','text', 'String','Last updated:', ...
        'Horizontal','right', 'Background',color);
    hLabelTime = uicontrol('Parent',hFig, 'Pos',[225,10,100,17], ...
        'Style','text', 'String','(not yet)', ...
        'Horizontal','left', 'Background',color);

    % Send the market-depth request to IQFeed using IQC
    contractParams = {'symbol',symbol}; % symbol='@ES#'
    callbacks = struct('Market_depth',@mktDepthCallbackFcn);
    IQC('marketdepth', contractParams{:}, 'processFunc',callbacks);
```

¹⁶² http://IQC.net/files/IQC_MktDepth.m or https://UndocumentedMatlab.com/IQC/files/IQC_MktDepth.m

```
% Figure close callback function - stop market-depth streaming
function figClosedCallback(hFig, ~)
    % Delete figure window and stop any pending data streaming
    delete(hFig);
    IQC('marketdepth', contractParams{:}, 'numofevents',0);
end % figClosedCallback

% Start/stop button callback function
function buttonCallback(hButton, ~)
    currentString = get(hButton, 'String');
    if strcmp(currentString, 'Start')
        set(hButton, 'String', 'Stop');
    else
        set(hButton, 'String', 'Start');
    end
end % buttonCallback

% Callback functions to handle IQFeed Market Depth update events
function mktDepthCallbackFcn(~, eventData)

    % Ensure that it's the correct MktDepth event
    allMsgParts = strsplit(eventData.MessageString, ',');
    allMsgParts(strcmpi(allMsgParts, 'T')) = {true};
    allMsgParts(strcmpi(allMsgParts, 'F')) = {false};
    if strcmp(eventData.MessagePort, 'Level2') && ...
        strcmp(allMsgParts{2}, symbol)

        % These are the field names of the IQFeed messages
        inputParams = {'Intro', 'Symbol', 'MMID', ...
            'Bid', 'Ask', 'BidSize', 'AskSize', ...
            'BidTime', 'Date', 'ConditionCode', ...
            'AskTime', 'BidInfoValid', ...
            'AskInfoValid', 'EndOfMsgGroup'};

        % Get the updated data row
        % Note: Java indices start at 0, Matlab starts at 1
        mmid = allMsgParts(strcmpi(inputParams, 'MMID'));
        row = sscanf(mmid, '%*c%*c%d');

        % Get the size & price data fields from the event's data
        bidValid = allMsgParts(strcmpi(inputParams, 'BidInfoValid'));
        askValid = allMsgParts(strcmpi(inputParams, 'AskInfoValid'));
        bidTime = allMsgParts(strcmpi(inputParams, 'BidTime'));
        askTime = allMsgParts(strcmpi(inputParams, 'AskTime'));
        bidSize = allMsgParts(strcmpi(inputParams, 'BidSize'));
        askSize = allMsgParts(strcmpi(inputParams, 'AskSize'));
        bidPrice = allMsgParts(strcmpi(inputParams, 'Bid'));
        askPrice = allMsgParts(strcmpi(inputParams, 'Ask'));
        thisRowsData = {askValid, askTime, askSize, askPrice, ...
            bidPrice, bidSize, bidTime, bidValid};
        depthData(row, :) = thisRowsData;

        % Update the GUI if more than 0.5 secs have passed and
        % the <Stop> button was not pressed
        if ~isvalid(hButton), return, end
        isStopped = strcmp(get(hButton, 'String'), 'Start');
        if now - lastUpdateTime > GUI_refresh_period && ~isStopped
            set(hTable, 'Data', depthData);
            set(hLabelTime, 'String', datestr(now, 'HH:MM:SS'));
            lastUpdateTime = now;
        end
    end
end % mktDepthCallbackFcn
end % IQC_MktDepth
```

IQML market-depth example

	Ask valid	Ask time	Ask size	Ask price	Bid price	Bid size	Bid time	Bid valid
1	<input checked="" type="checkbox"/>	09:56:49.192521	21	2704.75	2704.5	193	09:56:49.192176	<input checked="" type="checkbox"/>
2	<input checked="" type="checkbox"/>	09:56:49.192476	179	2705.	2704.25	271	09:56:49.192176	<input checked="" type="checkbox"/>
3	<input checked="" type="checkbox"/>	09:56:49.192361	176	2705.25	2704.	496	09:56:49.192176	<input checked="" type="checkbox"/>
4	<input checked="" type="checkbox"/>	09:56:49.192205	666	2705.5	2703.75	399	09:56:49.192176	<input checked="" type="checkbox"/>
5	<input checked="" type="checkbox"/>	09:56:49.192205	224	2705.75	2703.5	390	09:56:49.192176	<input checked="" type="checkbox"/>
6	<input checked="" type="checkbox"/>	09:56:49.192205	295	2706.	2703.25	424	09:56:49.192176	<input checked="" type="checkbox"/>
7	<input checked="" type="checkbox"/>	09:56:49.192205	340	2706.25	2703.	480	09:56:49.192426	<input checked="" type="checkbox"/>
8	<input checked="" type="checkbox"/>	09:56:49.192451	430	2706.5	2702.75	331	09:56:49.192176	<input checked="" type="checkbox"/>
9	<input checked="" type="checkbox"/>	09:56:49.192205	373	2706.75	2702.5	350	09:56:49.192176	<input checked="" type="checkbox"/>
10	<input checked="" type="checkbox"/>	09:56:49.192205	333	2707.	2702.25	349	09:56:49.192176	<input checked="" type="checkbox"/>

Last updated: 16:56:53

11 Alerts

11.1 General Usage

In cases where certain events in streaming data are of interest to the user, *IQC* can generate alerts of these events as they arrive from IQFeed. The user can define the event data type, the trigger condition, and the type of alert to generate when the condition is met. For example, users may configure an alert on quotes, such that when a symbol's bid price is higher than some threshold, an email will be sent.

Each alert contains 3 components:

- Data type – quote, intervalbar, regional or news
- Trigger – a condition (typically a comparison between a field and a value)
- Action – what *IQC* should do when the trigger condition is met

Alerts are created using the 'alert' action. Each new alert is assigned a unique numeric ID. Using this ID, users can query, delete or edit the alert after it was created.

The following parameters affect the alerts. Detailed explanations and usage examples are listed in the following sections.

Parameter	Data type	Default	Description
Symbol or Symbols ¹⁶³	colon or comma-delimited string or cell-array of strings	(none)	Limits the alert to the specified symbols and meta-tags only. Examples: <ul style="list-style-type: none"> • 'IBM' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} Optional parameter for news alerts; mandatory for quote/intervalbar alerts
Trigger	string describing the alert trigger	(none) – must be defined for new alerts!	A string composed of the data type, triggering parameter, trigger operator and triggering value, separated by spaces. Examples: <ul style="list-style-type: none"> • 'quote bid >= 102.60' • 'intervalbar close < 80' • 'news text contains IPO'
AlertAction	string, function handle, or callback cell array	(none) – must be defined for new alerts!	Type of alert to generate. Options: ¹⁶⁴ <ul style="list-style-type: none"> • 'display' • 'popup' • 'email' (requires specifying the EmailRecipients parameter) • @myCallbackFcn • {@myFcn, data1, data2, ...}

¹⁶³ In *IQC*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

¹⁶⁴ Note the performance implications that are discussed in §3.6 and §10.2

Parameter	Data type	Default	Description
NumOfEvents	integer	1	Maximal # of times to be alerted of the defined event. NumOfEvents = -1 returns a list of all existing alerts.
StartStream	logical (true/false)	false	If false (default), data streaming needs to be started by the user in a separate command. If true and relevant data streaming is not currently active, <i>IQC</i> starts the data streaming automatically (see §11.2).
AlertID	integer (scalar or array)	[] (empty array)	Unique ID generated and returned by <i>IQC</i> when new alert is defined. AlertID is relevant (and mandatory) only for querying, editing or deletion of existing alerts. See §11.3 and §11.4.
GetStory	logical (true/false)	true	If true (default), the full story text is fetched and reported with each news alert via email/callback; if false, only headline data will be reported. GetStory is relevant only for news alerts with AlertAction ='email' or callback.
EmailRecipients	comma-delimited string or cell-array of strings	" (empty string)	Email addresses to which email alerts will be sent. This parameter is relevant (and mandatory) only for email alerts. Examples: <ul style="list-style-type: none"> • 'john@doe.com' • 'john@doe.com, jane@doe.com' • {'john@doe.com', 'jane@doe.com'}
SmtplibEmail	string	'IQC.alerts@gmail.com'	SMTP e-mail address from which alert emails will be sent. This parameter is relevant only for specifying a non-default email sender. SmtplibEmail only needs to be set once, and is used by all future <i>IQC</i> alert events.
SmtplibServer	string	(none)	SMTP server that will send alert emails. This parameter is relevant only for specifying a non-default email sender. SmtplibServer only needs to be set once, and is used by all future <i>IQC</i> alert events.
SmtplibPassword	string	(confidential)	Password of the sender's e-mail account. This parameter is relevant only for specifying a non-default email sender. SmtplibPassword only needs to be set once, and is used by all future <i>IQC</i> alert events.



Note: Alerts are only available in the Professional *IQC* license.

11.2 Alert Configuration

Alerts can be configured by the user using the 'alert' action, using the properties in the table above. Users can configure the data type, event trigger, maximal number of alert reports, and the type of alert to generate (email, pop-up message, etc.). For email alerts, users can also specify the recipients and the sender email account.

The **Trigger** parameter is the most important input, and is unique to the `alert` action. It is a string describing the alert trigger event, so it is very important that it be composed properly. The **Trigger** string has 4 elements:

1. Data type ('quote', 'intervalbar', 'news' or 'regional')
2. Trigger field: case-insensitive name of a field in the `latestData` struct of the source data specified by the Data type (see §6.1, §6.3). For example: 'bid', 'ask', 'total_volume', 'Most_Recent_Trade', 'intervalVolume', 'text', etc.
3. Trigger operator ('>', '>=', '<', '<=', '=', or 'contains').¹⁶⁵
 - '>', '<', '>=', '<=' are only relevant for non-news alerts (but not for news)
 - '=' and 'contains' are relevant for all alert types (including news)
4. Trigger value: either a scalar number (for a '>', '>=', '<', '<=', or '=' operator) or a string (for a '=' or 'contains' operator).

For example:

```
alertId = IQC('alert', 'Symbol', 'IBM', 'Trigger', 'quote ask < 145.70', ...);
alertId = IQC('alert', 'Symbol', 'IBM', 'Trigger', 'quote Total_Volume >= 10', ...);
alertId = IQC('alert', 'Symbol', 'IBM', 'Trigger', 'news text contains IPO', ...);
```

By default, alerts are only triggered and reported once. This can be changed by setting the **NumOfEvents** parameter to an integer value. For example, the following alert will be reported up to 5 times, and will then be deleted from the list of alerts:

```
alertId = IQC('alert', 'Symbol', 'IBM', ..., 'NumOfEvents', 5);
```



IQC does NOT automatically start streaming data when alerts are defined. This enables users to start and stop streaming data at will, and the alerts will only be evaluated when streaming data messages arrive from IQFeed.

Note: if you use a Data type of 'quote', then depending on the setting of the **Fields** parameter in your latest quotes query (§4.1), the requested alert Trigger field might not exist in the streaming quotes, causing the alert to become ineffective. *IQC* does **not** automatically update **Fields** with the requested Trigger field. When such a case is detected, a warning message will be presented:

```
Warning: Field 'VWAP' is not currently included in your quotes Fields
parameter, making the requested alert useless
```

It is sometimes convenient to start streaming immediately when the alert is created. This can be done by setting the **StartStream** parameter (default `false`). Setting a value of `true` starts the streaming for the corresponding data type (e.g., streaming quotes for a symbol) automatically, unless the streaming is already active.

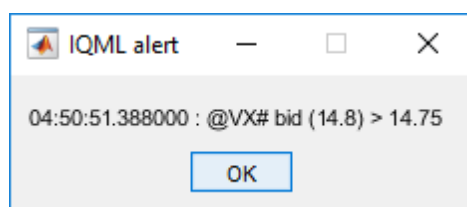
¹⁶⁵ Additional trigger operators may be added in future *IQC* releases.

Note that with **StartStream=true** the streaming is started automatically, using the default parameters. If you wish to control the streaming parameters (for example, **NumOfEvents** or **DataType**), leave **StartStream** in its default `false` value, and start the streaming in a separate *IQC* command.

The **AlertAction** defines the action to be performed when a triggering event is detected (i.e., when the trigger condition is met). There are 4 possible **AlertAction** values: 'popup', 'display', 'email', and callback (note the performance discussion in §3.6, §10.2):

1. 'Popup' announces the triggered event in a pop-up a message-box:

```
alertId = IQC('alert', 'Symbol', '@VX#', 'Trigger', 'quote bid > 14.75', ...
             'AlertAction', 'popup');
```



2. 'Display' announces the event in Matlab's console (Command Window):

```
alertId = IQC('alert', 'Symbol', '@VX#', 'Trigger', 'quote bid > 14.75', ...
             'AlertAction', 'display');
```

```
04:50:11.099000 IQC alert: @VX# bid (14.8) > 14.75
```

Or, as another example of regional update alert:

```
alertId = IQC('alert', 'Symbol', 'IBM', 'AlertAction', 'display', ...
             'Trigger', 'regional regionalbid > 140');
```

```
20180524 16:57:13.689 IQC alert: IBM regionalbid (143.75) > 140
```

3. 'Email' – an email with the alert event's details will be sent to the specified **EmailRecipients**, a mandatory parameter for email alerts. **EmailRecipients** must be set as a comma/semi-colon/colon delimited string, or a cell array of email addresses; it cannot be left empty.

For example, the following alert will send an email to two email recipients:

```
alertId = IQC('alert', 'Symbol', '@VX#', 'Trigger', 'quote bid > 14.75', ...
             'AlertAction', 'email', ...
             'EmailRecipients', {'john@a.com', 'jane@b.com'});
```

which results in an email similar to this:

```
From: IQC.alerts@gmail.com
Subject: IQC alert: @VX# bid (14.8) > 14.75
Body:
Symbol: '
Most_Recent_Trade: 14.82
Most_Recent_Trade_Size: 10
Most_Recent_Trade_Time: '08:40:02.926510'
Most_Recent_Trade_Market_Center: 32
Total_Volume: 6890
```

Bid: 14.8
...

or similarly, in the case of a news alert:

```

From: IQC.alerts@gmail.com
Subject: IQC alert: United Technologies Plans To Hire 35,000 People, Make $15 B... (RTB)
Body:
    ID: 22017029634
    Symbols: {'UTX'}
    Text: '09:31 Wednesday, May 23, 2018. (RTTNews) - United Technologies Plans To Hire 35,000 People, Make $15 Bln Investment In U.S. Over Next 5 Years For comments and feedback: contact editorial@rttnews.com. Copyright(c) 2018 RTTNews.com. All Rights Reserved'
```

For news alerts, the full story text is fetched by default. It is possible to skip fetching the full story by setting **GetStory** to `false`. This speeds up processing by skipping the news-fetch query, and reports only the headline information:

```

From: IQC.alerts@gmail.com
Subject: IQC alert: United Technologies Plans To Hire 35,000 People, Make $15 B... (RTB)
Body:
    Source: 'RTB'
    ID: 22017029634
    Symbols: {'UTX'}
    Timestamp: '20180523 093143'
    Text: 'United Technologies Plans To Hire 35,000 People, Make $15 B...'
```

As noted, **EmailRecipients** can be specified in various manners. For example, all the following are equivalent:

```

'EmailRecipients', 'john@a.com,jane@b.com'
'EmailRecipients', 'john@a.com;jane@b.com'
'EmailRecipients',{'john@a.com','jane@b.com'}
```

Alert emails are sent from an *IQC* email address (IQC.alerts@gmail.com) by default. To send the alert emails from another sender (for example, a corporate email account), specify the **SmtplibEmail**, **SmtplibServer** and **SmtplibPassword**.¹⁶⁶ These parameters are saved in your local machine's Matlab settings, and will be used by all future *IQC* email alerts (even after you restart the computer), so you only need to set them once. For example:

```

alertId = IQC('alert', 'Symbol', 'GOOG', 'Trigger', 'quote ask < 1090', ...
    'AlertAction', 'email', 'Recipients', 'JohnDoe@gmail.com', ...
    'SmtplibServer', 'smtp.gmail.com', ...
    'SmtplibEmail', 'senderEmail@gmail.com', ...
    'SmtplibPassword', 'mypassword123');
```

On modern smartphones, text (SMS) messages have generally been replaced with email push notifications. Still, for some users text alerts may be useful. Some mobile operators enable users to receive text messages by sending them to a specially-formed email address.¹⁶⁷ For example, to send a text message alert to T-Mobile number 123-456-7890 in the USA, simply email the alert to

¹⁶⁶ The SMTP port is automatically assumed to be 465. If you use Google's mail server (smtp.gmail.com), the account must allow access from "less secure apps" (<https://myaccount.google.com/lesssecureapps>). Note that anti-virus or firewall software may possibly block the outgoing emails (in such cases you may see a "PKIX path building" error).

¹⁶⁷ https://en.wikipedia.org/wiki/SMS_gateway#Email_clients

1234567890@tmomail.net. To receive alerts via such text messages, you just need to determine your mobile carrier's email gateway for SMS messages, and set **EmailRecipients** accordingly. Note that carrier charges might apply.

4. **Callback:** a personalized callback function for an event can be specified using a Matlab function handle. For example:

```
alertId= IQC('alert','Symbol','GOOG','Trigger','...'AlertAction',@myFunc);
```

The callback function (`myFunc` in this example) should accept two or more inputs, as customary for Matlab callbacks.¹⁶⁸

```
function myFunc(alertObject, eventData)
```

- `alertObject` a struct with the alert's configuration (see §11.3)
- `eventData` a struct with the triggered event's local time (in Matlab datenum format) and the trigger data.

For example, for quote data alerts, `eventData` might look like this:

```
>> eventData =
    triggerTime: 737202.663148947
    triggerData: [1x1 struct]

>> eventData.triggerData
ans =
                                     Symbol: 'GOOG'
                Most_Recent_Trade: 1083
    Most_Recent_Trade_Size: 30
    Most_Recent_Trade_Time: '08:54:53.159809'
    Most_Recent_Trade_Market_Center: 11
                Total_Volume: 1957
                ...
```

To specify additional input parameters to your callback function, set **AlertAction** to a cell array in which the first cell is the function handle and the rest are additional inputs. For example:

```
callback = {@myFunc, data1, data2};
alertId= IQC('alert','Symbol','GOOG','Trigger','...'AlertAction',callback);

function myFunc(alertObject, eventData, data1, data2)
    ... % data processing done here
end
```

¹⁶⁸ https://www.mathworks.com/help/matlab/creating_plots/callback-definition.html;
https://www.mathworks.com/help/matlab/creating_guis/write-callbacks-using-the-programmatic-workflow.html#f16-1001315

11.3 Alerts Query

IQC can be queried for the list of all existing alerts, or just a single specific alert. Alerts are returned in this case as Matlab structs containing the alerts' specifications.

Specific alerts may be queried by specifying their unique **AlertID** (which was returned by the command that created the alert), and setting **NumOfEvents** to -1:

```
>> alertID = IQC('alert', 'Symbol', 'IBM', 'Trigger', 'quote bid > 200', ...);
>> alert = IQC('alert', 'AlertID', alertID, 'NumOfEvents', -1)
alert =
    struct with fields:
        AlertID: 22120136109
        isActive: 1
        DataType: 'quote'
        Trigger: 'bid > 200'
        TriggerType: 'bid'
        TriggerOp: '>'
        TriggerValue: '200'
        Symbol: {'IBM'}
        AlertAction: 'popup'
        EmailRecipients: {}
        EventsProcessed: 0
        EventsToProcess: 1
        LatestValue: []
```

The **AlertID** parameter can be an array of alert IDs, resulting in an array of structs.

To retrieve the list of all the existing alerts, simply set **NumOfEvents** to -1, without specifying the **AlertID** parameter:

```
>> allAlerts = IQC('alert', 'NumOfEvents', -1)
allAlerts =
    3x1 struct array with fields:
        AlertID
        isActive
        DataType
        Trigger
        TriggerType
        TriggerOp
        TriggerValue
        Symbol
        AlertAction
        EmailRecipients
        EventsProcessed
        EventsToProcess
        LatestValue
```

11.4 Alert Editing or Deletion

An existing alert can be edited or deleted by specifying its **AlertID**:

To delete an alert, set **NumOfEvents** to 0 as follows:

```
IQC('alert', 'AlertID', alertID, 'NumOfEvents', 0);
```

To update/edit an alert, specify **AlertID** with one or more of the alert configuration parameters: **Symbols**, **Trigger**, **AlertAction**, **EmailRecipients**, **NumOfEvents** (>1).

```
IQC('alert', 'AlertID', alertID, 'AlertAction', 'email', 'EmailRecipients', 'john@a.com');
```

As above, the **AlertID** input can be an array of IDs, affecting multiple alerts at once.

12 Messages and logging

12.1 IQC messages

To display detailed information on *IQC* requests and IQFeed messages, set *IQC*'s **Debug** parameter to 1 or `true` (default=0). *IQC* will then display in the Matlab console (Command Window) additional information that may help diagnose problems.

For example, setting **Debug** to 1 (or `true`) displays the outgoing commands from *IQC* to IQFeed (“=>”), and incoming messages from IQFeed to *IQC* (“<="), along with the message's local timestamp and port channel:¹⁶⁹

```
>> data = IQC('news' , 'DataType', 'headlines', 'MaxItems', 4, 'debug', 1)
=> 20180401 15:14:00.010 (Lookup) NHL,,,:t,5,,
<= 20180401 15:14:01.082 (Lookup) N,CPR,21998204468,,20180401080059,
Following Is a Test Release
<= 20180401 15:14:01.086 (Lookup) N,RTI,10134529069,,20180401080029,
Quarterly Corporate Earnings (04/01/18)
<= 20180401 15:14:01.092 (Lookup) N,CPR,21998201110,,20180401073059,
Following Is a Test Release
<= 20180401 15:14:01.098 (Lookup) N,CPR,21998197500,,20180401070059,
April 1 Alert: Introducing, Duty Not Free: Pay-as-you-go toilet time
<= 20180401 15:14:01.107 (Lookup) !ENDMSG!

>> data = IQC('quotes', 'symbol', 'FB', 'debug', 1)
=> 20180401 17:20:29.189 (Level1) wFB
<= 20180401 17:20:29.450 (Level1)
F,FB,5,29.1,50158000,195.3200,138.8100,195.3200,149.0200,0.0000,,,,,
,,,5.49,,2.52,12,,FACEBOOK,FB,47.600,0.63,,48563.0,3760.0,12/31/2017
,,2905001,,,,,14,4,7375,36.25,1,21,02/01/2018,04/11/2017,02/01/2018,
03/26/2018,176.4600,,,,,519190,,,
<= 20180401 17:20:29.462 (Level1)
P,FB,160.0500,50000,19:59:56.263577,11,0,160.0500,4600,160.0600,200,,
,,159.7900,Cbacv,8801
=> 20180401 17:20:29.471 (Level1) rFB
```

In order to log such messages in a text file, you can use IQFeed's built-in logging facility, as described below (§12.2).

In certain cases, *IQC* reports messages as red error messages on the Matlab console. Such messages can be handled by analyzing *IQC*'s second (optional) output argument, which is typically an empty string, except when an error is reported:

```
>> [data, errorMsg] = IQC('quotes', 'Symbol', 'IBM', 'Timeout', 0.1)
IQC timeout: either IQFeed has no data for this query, or the Timeout
parameter should be set to a value larger than 0.1
data =
[]
errorMsg =
'IQC timeout: either IQFeed has no data for this query, or the Timeout
parameter should be set to a value larger than 0.1'
```

Users can control whether such error messages from IQFeed should raise a Matlab error (exception) in blocking (non-streaming) mode, using the **RaiseErrorMsgs** parameter (default: `true`).

```
>> [data, errorMsg] = IQC('quotes', 'Symbol', 'IBM', 'RaiseErrorMsgs', false);
```

¹⁶⁹ Periodic IQFeed timestamp and client-stats messages (once every second) are not displayed, even when **Debug** is 1 or `true`. For a description of the various port channels used by IQFeed, see §9.1.

In addition to IQFeed messages, your program must handle cases of *IQC* errors. In most cases, these are due to invalid *IQC* input parameters (an invalid action or parameter name, or an invalid parameter value). Errors might also happen due to network problems, or even an internal bug due to an unhandled edge-case situation.

To trap and handle such programmatic exceptions, wrap your calls to *IQC* within a try-catch block, as follows:

```
try
    data = IQC('action','query', ... );
catch
    % process the exception here
end
```

Try-catch blocks have very small performance or memory overhead and are a very effective way to handle programmatic errors. We recommend that you use them in your program, not just to wrap *IQC* calls but also for other processing tasks. I/O sections in particular (reading/writing files) are prone to errors, so they are prime candidates for such exception handling. The same applies for code that handles user inputs (we can never really be too sure what invalid junk a user might enter in there, can we?).

Very common causes of errors when using *IQC* are relying on default parameter values, and specifying numeric parameter values within string quotes (e.g., '1' rather than 1). Users of *IQC* should take extra precaution in their programs to ensure that these common mistakes do not occur. See discussion in §3.4.

Matlab “out of memory” errors might occur when receiving and storing a huge amount of streaming or historic data. They can be fixed by running *IQC* on a computer having more memory, or by reducing the amount of stored data.¹⁷⁰

Java memory errors are recognized by the message `java.lang.OutOfMemoryError: Java heap space`. They can be solved by running Matlab with more allocated Java heap memory than the default value of 64MB or 128MB (depending on Matlab release). This value can be increased in Matlab’s preferences, or via a *java.opts* file.¹⁷¹

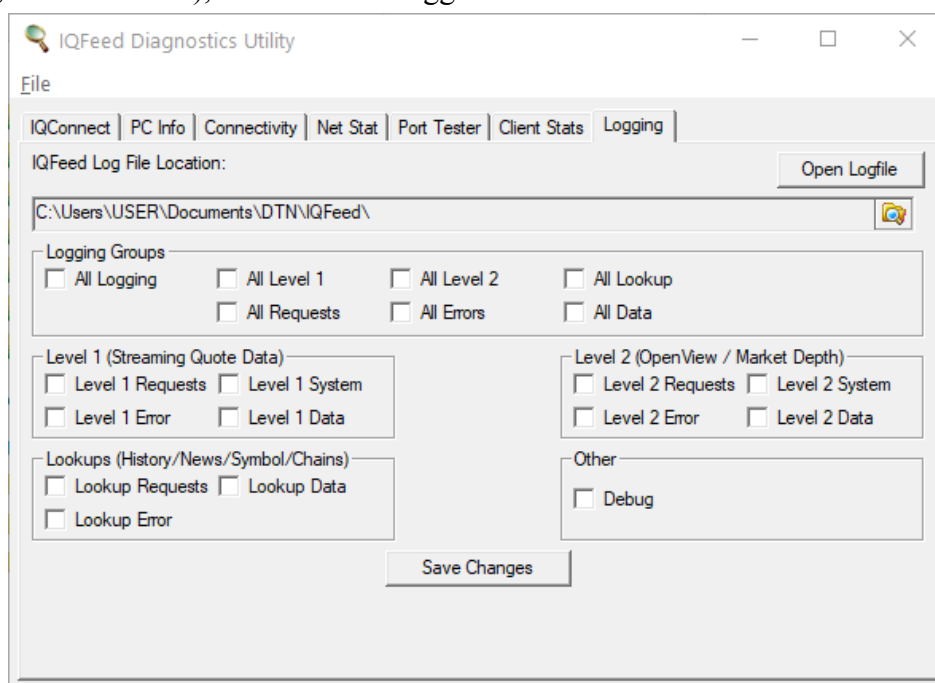
¹⁷⁰ Also see: http://www.mathworks.com/help/matlab/matlab_prog/resolving-out-of-memory-errors.html

¹⁷¹ <https://www.mathworks.com/matlabcentral/answers/92813-how-do-i-increase-the-heap-space-for-the-java-vm-in-matlab-6-0-r12-and-later-versions>

12.2 IQFeed logging

IQFeed requests, messages and events (e.g. connections/disconnections), are logged in IQFeed's log. This is a text file called "*IQConnect.txt*" in the *\DTN\IQFeed* subfolder of "My Documents", e.g. *C:\Users\<xyz>\Documents\DTN\IQFeed\IQConnectLog.txt* (replace *<xyz>* with the actual user name in your computer).

Using IQFeed's Diagnostic Utility, which is installed as part of IQFeed's client installation, you can control the log file's folder path (but not its *IQConnectLog.txt* name, which is fixed), as well as the logged details:



Running `IQC('log')` without any parameters will return a Matlab struct containing the current log settings (the struct fields are explained below):¹⁷²

```
>> data = IQC('log')
data =
    struct with fields:
        logDetails: [1 0 0 0 0]
        logFile: 'C:\Users\Yair\Documents\DTN\IQFeed\IQConnectLog.txt'
```

In addition to interactively using the Diagnostic Utility to modify the log file path and logging details, you can also control them programmatically, via *IQC*'s 'log' action:

We can modify the log file's folder path using the **Path** parameter, for example:

```
IQC('log', 'path', 'C:\My Programs\Logs\')
```

Note: **Path** sets the log file's folder – not its name. As mentioned above, IQFeed's log file name is fixed (*IQConnectLog.txt*) and cannot be modified, only its folder can be changed. As a safety measure, *IQC* will complain if the specified **Path** does not exist:

```
>> IQC('log', 'path', 'C:\No\Such\Folder')
Error using IQC
Bad log path specified: "C:\No\Such\Folder" is not an existing folder
```

¹⁷² Note: the `IQC('log')` feature uses IQFeed's Windows registry. In Mac/Linux, `IQC('log')` will naturally work only when Matlab runs under Parallels/Wine, not in native mode.

When setting the log **Path**, the *IQC* command will return a Matlab struct with the log's old filename (prior folder name and the fixed name *IQConnect.txt*). You can use this to restore the original log path after temporarily redirecting logging to a different folder:

```
data = IQC('log', 'path', newPath);
oldPath = fileparts(data.logFile); % strip out the IQConnectLog.txt file name
... % logging in this section is temporarily redirected to newPath
IQC('log', 'path', oldPath); % restore the original log file's path
```

Note that changing the log file's **Path** does not affect a currently-running IQFeed session, only subsequent sessions (IQFeed's *IQConnect* client reads the log **Path** when it launches). If you need to modify the **Path** while IQFeed is running, you can temporarily reconnect IQFeed (see §9.1) to ensure that it loads the new **Path** value.



Be careful when changing log **Path**. Its default location is set to the user's *MyDocs* folder since by default *IQConnect.exe* launches under the user's permission group and all users have full access rights to files in their user folder. Ensure that your user account has appropriate permissions in the chosen folder. Also ensure that the folder is not synced (to OneDrive, Dropbox etc.), which could drastically reduce system performance.

Also keep in mind that there is only one log file for all applications that use IQFeed on the computer. If your program might run alongside other software that uses IQFeed, it is recommended that you do not change the logging location from the default.

We can specify the logging details using the **Details** parameter, which accepts an array of up to 5 numeric or logical values. These values correspond to Administrative,¹⁷³ Level 1, Level 2, Lookup,¹⁷⁴ and Debug.¹⁷⁵ By default, IQFeed logs only Admin requests and messages, which corresponds to **Details**=[1,0,0,0,0] or simply 1 (extra zeros are assumed, so 1 means the same as [1,0,0] or [1,0,0,0,0]). For example, to also log Level 1 and Lookup messages (but not Level 2 or Debug), set **Details** to [1,1,0,1]:

```
>> data = IQC('log', 'details', [1 1 0 1])
data =
    struct with fields:
        logDetails: [1 0 0 0 0]
        logFile: 'C:\Users\Yair\Documents\DTN\IQFeed\IQConnectLog.txt'
```

In this example, note how *IQC* returned the previous **Details** setting, prior to its change. A subsequent call to *IQC*('log') will verify that the **Details** change was indeed made:

```
>> data = IQC('log')
data =
    struct with fields:
        logDetails: [1 1 0 1 0]
        logFile: 'C:\Users\Yair\Documents\DTN\IQFeed\IQConnectLog.txt'
```



Note: IQFeed's log file can become VERY large VERY fast and potentially reduce system performance if left unattended. It is intended to be used for troubleshooting purposes only and not on a constant basis, and should be used very carefully. This is especially true if you log streaming data, large historic data, and/or Debug data.

Also note that the logging level **Details** are stored in the computer's registry and persist across different sessions of Matlab and IQFeed. So after you have set detailed logging and no longer need it, it is good practice to immediately set **Details** back to [1].

¹⁷³ Admin includes connection/disconnection and other non-data requests and messages

¹⁷⁴ Lookup includes history requests/data (§5), news (§7), and symbols/chains lookup (§8)

¹⁷⁵ IQFeed's Debug data provides even more granular logging to aid in troubleshooting issues than the other four log types. This is not related to the **Debug** parameter that was discussed in §12.1.

Note that unlike **Path** (which does not affect a currently-running IQFeed session), setting the logging **Details** DOES indeed affect the current session; no restart of the IQFeed client is required for the new **Details** setting to take effect.

In addition to setting log **Path** and **Details**, we can also use the 'log' action for several special requests: The **CopyTo** parameter copies the current log file into a specified folder/path. We can also specify the filename, since it is just a copy of the live log file. Note: if the specified target file already exists, it will be overwritten.

```
>> IQC('log', 'copyTo', 'C:\My Programs\Logs\log.txt'); % filename=log.txt
>> IQC('log', 'copyTo', 'C:\My Programs\Logs\'); % filename=IQConnectLog.txt
```

We can use the **DoThis** parameter to reset (empty) the live log file, or to display it in an external editor (the editor displays a log snapshot, it does not automatically refresh):

```
>> IQC('log', 'doThis', 'reset'); % reset (empty) the live log file
>> IQC('log', 'doThis', 'display'); % display live log file in external editor
```

Note: Both **CopyTo** and **DoThis** only affect the log file at the time of the request, not continuously. In other words, **CopyTo** copies a snapshot of the live log file as of the time of request; **DoThis** 'reset' (or 'clear' or 'empty') does a one-time reset of the log; and **DoThis** 'display' (or 'show') displays the current log file snapshot (the editor's refresh of this file is not automatic).

As with other *IQC* actions, we can combine different parameters in a single *IQC* command. For example:

```
IQC('log', 'details', [1 1 0 1], 'path', 'C:\Programs\Logs\', 'doThis', 'reset');
```

Here is a summary of the *IQC* parameters that affect IQFeed's internal logging:

Parameter	Data type	Default	Description
Path	string	(last-used log folder path; initially set to <i>My Documents\DTN\IQFeed</i>)	Path of the folder in which the <i>IQConnect.txt</i> live log file is to be stored/updated.
Details	numeric or logical array	(last-used log details setting; initially set to [1,0,0,0,0] meaning only Admin msgs are logged)	Array of up to 5 numeric/logical values, corresponding to Admin, Level1, Level2, Lookup, and Debug requests/messages. A value of 1 (or true) indicates that requests/messages belonging to the corresponding group should be logged; a value of 0 (or false) indicates that they should not be logged.
CopyTo	string	" (empty string)	Path of folder or file in which a snapshot copy of the live log file is to be placed, overwriting existing file if such a file already exists.
DoThis	string	" (empty string)	One of the following string values: <ul style="list-style-type: none"> 'display' or 'show' – display the log file 'reset' or 'clear' – empty the live log file

13 Frequently-asked questions (FAQ)

1. Can IQC be used with other data-feed providers?

IQC only connects to DTN IQFeed. It can be adapted for other data providers, but some development is obviously required since other providers have different APIs. Email us to see if we can help.

2. Does IQC impose limitations on historical data or streaming quotes?

No – *IQC* does not impose any limitations. However, IQFeed’s servers do impose limitations on the frequency of the requests and the amount of returned data. These limitations depend on your specific IQFeed subscription. For example, your account might be limited to some maximum number concurrently-streaming (“watched”) symbols. These limitations are imposed by the IQFeed server on your account; *IQC* supports whatever entitlements your IQFeed account has, it does not limit the information in any manner.

3. Can I see a demo of IQC?

Yes – you are welcome to download a fully-functional trial version of *IQC*, to try the product at no risk for 30 days.

4. How does IQC compare to alternative products?

We believe that of all the currently available alternatives for connecting Matlab to IQFeed, *IQC* provides by far the best functionality, value and cost-effectiveness. You are most welcome to test this yourself, using *IQC*’s free trial. Several traders have reviewed *IQC* and claim that it is the best Matlab-to-IQFeed connector.¹⁷⁶

5. Does IQC come with an IQFeed or market subscription?

No – *IQC* connects to an existing IQFeed account. You will need to purchase the IQFeed and market subscriptions separately from DTN.

6. Does IQC send you any information?

No – *IQC* only communicates with IQFeed. The only communication that is done with *IQC*’s server is a verification of the license activation (a single hash-code).

7. How can I be sure IQC does not contain bugs that will affect my trades?

The product is rigorously tested, but there is no 100% guarantee. To date, nothing major has been reported. *IQC* is rock solid - a very stable and robust product.

8. Is IQC being maintained? supported?

Yes, actively. Features and improvements are added on a regular basis, and we support the users personally. You can see the list of ongoing improvements in *IQC*’s change-log, listed in Appendix B of the *IQC* User Guide (this document). You can see the very latest updates in the online version of this guide.¹⁷⁷

¹⁷⁷ http://IQC.net/files/IQC_User_Guide.pdf or https://UndocumentedMatlab.com/IQC/files/IQC_User_Guide.pdf

9. I saw a nice new feature in the online User Guide – can I get it?

Once you install *IQC*, you will be notified in the Matlab console (Command Window) whenever a new version is available. You can always update your installation to the latest version, using a variety of means, as explained in §2.4.

10. What happens when the license term is over?

A short time before your license term is over, you will start to see a notification message in your Matlab console (Command Window) alerting you about this:

```
*** Your IQC license will expire in 3 days (10-Mar-2018) .  
*** To extend your license please email info@IQC.net
```

This message will only appear during the initial connection to IQFeed, so it will not affect your regular trading session. When the license term is over, *IQC* will stop working. You can always renew or extend your license using the payment link generated by IQC at the Matlab prompt.

11. Can I transfer my IQC license to another computer?

Yes, simply email us and we will make the activation switch for you. At any one time, each *IQC* license will only be activated on a single computer, unless you purchase a site license. You can make up to 3 license activations per year at no extra cost; additional switches will incur a handling fee.

12. I have a laptop and desktop – can I use IQC on both?

Yes, but you will need to purchase two separate *IQC* licenses. *IQC*'s license is tied to a specific computer, unless you purchase a site license.

13. Can IQC be compiled and deployed?

Yes, *IQC* can be compiled using the Matlab Compiler. Each computer running a compiled *IQC* requires an *IQC* license (just like a non-compiled *IQC* that runs in Matlab), unless you get a group license (Site, Deployment, or Development). In other words, for your deployed computers you have a choice of either buying individual licenses (separately for each deployed computer), or a group license which does not require dedicated license activations. If you wish to deploy *IQC* on a large scale for multiple end-user computers, contact us to discuss alternatives.

14. Is IQC provided in source-code format?

IQC is provided in encrypted binary form, like any other commercial software. A source-code license is available for purchase, subject to signing a separate non-disclosure (NDA) agreement. The source-code version has no license fees and is not tied to any specific computer – you can install it on as many computers as you wish within your organization. Contact us for details. Also see related question #15 below.

15. Do you provide an escrow for IQC's source-code? Is the source code for sale?

Yes. There are two optional levels of escrow service that you can select:

1. At safe-keeping with a Wall-Street lawyer
2. Using NCC Group's¹⁷⁸ independent escrow service

Escrow services incur a non-negligible annual usage fee, but you may decide that it may be worth the optional extra cost to ensure business continuity.

Alternatively, a source-code license is available for purchase, subject to a separate non-disclosure (NDA) agreement. See related question #14 above.

Alternatively, purchasing a multi-year license will ensure independence of renewals, and a site license will avoid external activation checks during the license duration.

Contact us for details about any of these optional alternatives.

16. Is feature ABC available in IQC?

IQC supports the entire IQFeed API. This means that all the functionality that IQFeed exposes in its API, is available in *IQC* using an easy-to-use Matlab wrapper function. In addition to parametric queries, users can send IQFeed custom API commands (see §9.4) and then process the raw incoming IQFeed response (see §10). To check whether a specific feature is available in the IQFeed API (and by extension, in *IQC*), please refer to *IQC*'s User Guide (this document), IQFeed's online reference, or contact IQFeed customer service.

17. Can you add feature ABC in IQC for me?

We will be happy to do so, for a reasonable development fee that will be agreed with you in advance. After the development, this feature will be available to all others who purchase (or update) the latest version of *IQC*, at no extra cost. Contact us by email if you have such a request, to get a proposed quote.

18. Can you develop a trading strategy for me?

We will be happy to do so, for a reasonable development fee that will be agreed with you in advance. Unlike development of *IQC* features, strategy development will never be disclosed to others, and will not be integrated in *IQC*. It will be developed privately for you, and will be kept secret. See §15 for additional details. If you have such a request, contact us by email to get a proposed quote.

19. Does IQC include back-testing/charting/data analysis/algo-trading?

No. *IQC* is only used for communication with the IQFeed server (retrieving data from IQFeed servers) – it does not include any data analysis, charting or back-testing functionalities. Matlab is great at data analysis and visualization, so you can easily develop your own analysis programs in Matlab, using the data from *IQC*. We have extensive experience in developing complete backtesting and real-time trading applications - see §15 for additional details. We will be happy to either develop a new application based on your specifications, or to integrate *IQC* into your existing application, under a separate consulting contract.

¹⁷⁸ <http://nccgroup.com/en/our-services/software-escrow-and-verification/software-escrow>

14 Troubleshooting

Error	Description / solution	Section
<code>NullPointerException</code> <code>com.mathworks.jmi.bean.</code> <code>MatlabBeanInterface.-</code> <code>addCallback</code>	<i>IQC</i> cannot work properly unless its Java file (<i>IQC.jar</i>) is added to Matlab's static Java classpath. Contact us to solve the problem.	§2.1
<i>IQFeed</i> is not properly installed	<i>IQFeed</i> is not installed properly on the local computer so <i>IQC</i> cannot connect to it.	§2.1
<i>IQFeed</i> cannot be connected or started or: Cannot connect to <i>IQFeed</i>	<i>IQC</i> cannot connect to an active (running) <i>IQFeed</i> client process, nor start one. Try to start <i>IQFeed</i> 's client manually and then retry.	§2.1
<i>IQC</i> is not activated on this computer	Some component of your activated computer fingerprint has changed. Revert this change, or contact us to modify the activated fingerprint.	§2.2
Your <i>IQC</i> license will expire in 4 days (15-Jun-2018)	This is an alert on upcoming license expiration. It is not an error, and does not affect <i>IQC</i> 's operation. Contact us to extend your license.	§2.2
Your <i>IQC</i> license has expired on 1-Jun-2018	<i>IQC</i> 's license is limited in duration. When the license term expires, contact us to renew it.	§2.2
Cannot connect to <i>IQC.net</i> to validate your <i>IQC</i> license	<i>IQC</i> validates its license on the <i>IQC</i> server. Your internet connection may be down, or the domain (<i>IQC.net</i> , <i>undocumentedmatlab.com</i>) may be blocked by firewall (ask your IT to unblock it).	§2.2
Action 'xyz' is not [yet] supported	The specified action is not [yet] a valid <i>IQC</i> action, although it is planned for a future version.	§2.4
Unrecognized <i>IQC</i> action 'xyz'	The specified action is invalid in <i>IQC</i> . Refer to the User Guide for a list of acceptable actions.	§3.1
Missing parameter value: all parameters must have a value	No value was provided for the specified parameter. <i>IQC</i> parameters must be specified as name-value pairs that have both name and value.	§3.1
Value for parameter 'abc' should be a <xyz> data type	The specified parameter value provided in your <i>IQC</i> command has an incorrect data type. Refer to the User Guide for a list of acceptable values.	§3.1
Value for parameter 'abc' should be a scalar number	The specified parameter value must be a single scalar value, not a numeric array. Refer to the User Guide for a list of acceptable values.	§3.1
Warning: 'abc' is not a valid parameter for the 'xyz' action in <i>IQC</i>	The specified parameter name is not valid for the specified <i>IQC</i> action and is ignored. Refer to the User Guide for a list of acceptable parameters.	§3.1
The 'news' action is not available in your Standard license of <i>IQC</i>	The specified action is only available in the <i>IQC</i> Professional license and free trial. Contact us to upgrade your license to access this feature.	§3.4
Symbol 'XYZ' was not found	Either you have no permission to access this Symbol , or this symbol is unknown by <i>IQFeed</i> .	§3.4

Error	Description / solution	Section
(Missing digits in Matlab Command Window)	Matlab's display format is possibly set to "short" instead of "long".	§3.4
Undefined function 'struct2cell' for input arguments of type 'double'	An empty result was returned, and this cannot be converted into a Matlab cell-array.	§3.5
Error using struct2table (line 26) - S must be a scalar structure, or a structure array ...	An empty result was returned, and this cannot be converted into a Matlab table object.	§3.5
The Symbol parameter must be specified for an XYZ query when NumOfEvents>0	Queries that have NumOfEvents >0 must be specified with a non-empty Symbol/Symbols .	§4, §6
Warning: IQC timeout: only partial data is returned. Perhaps the Timeout parameter should be set to a value larger than 5	The query took longer than expected to return all the data; only partial results have arrived from IQFeed before the <i>IQC</i> timed-out. To get all results, set the Timeout parameter to a larger value or the NumOfEvents parameter to a smaller value.	§4.1, §4.3, §5.1, §7.2, §8.1
IQC timeout: either IQFeed has no data for this query, or the Timeout parameter should be set to a value larger than 5	The query took longer than expected to return any data from IQFeed before <i>IQC</i> timed-out. Try to set the Timeout parameter to a larger value.	§12.1
Date parameter value must be either a string (YYYYMMDD, YYYY-MM-DD or YYYY/MM/DD) or datenum	The date/time format of one or more of the query parameters is incorrect. Refer to the User Guide for a description of the acceptable formats.	§5
Symbol "XYZ" is not currently streaming	Start data streaming (by sending a query with NumOfEvents >0) before querying streamed data	§6
(<i>IQC</i> stops receiving IQFeed streaming data)	Try to actively disconnect and reconnect to IQFeed, or to restart the <i>IQConnect</i> application.	§9.1
Unable to connect to L2IP server. Error Code: 10065 Error Msg: A socket operation was attempted to an unreachable host. (or a similar variant)	<i>IQConnect</i> lost the connection to IQFeed's servers. <i>IQConnect</i> will automatically reconnect as soon as possible, and in most cases you can ignore this message. You can also try to actively reconnect to IQFeed, or to check your internet connection.	§9.1
Out of memory or: Maximum variable size allowed by the program is exceeded or: Requested array exceeds maximum array size preference	This Matlab error might occur when receiving huge amounts of streaming/historic data. Different Matlab releases display different messages having the same basic idea. Run <i>IQC</i> on a computer with more memory, or reduce the amount of stored/processed data.	§12.1
java.lang.OutOfMemory Error: Java heap space	Set Matlab to use a larger Java heap memory size than the default value. This can be set in Matlab's preferences, or via a <i>java.opts</i> file.	§12.1