IQML User Guide

Version 2.26

October 26, 2019

Fully compatible with:

Windows, Linux, Mac OS

DTN IQFeed 5.0 - 6.1

MATLAB R2008b - R2019b

http://IQML.net
https://UndocumentedMatlab.com/IQML

Undocumented Matlab

The engineering choice for professional Matlab solutions

Table of Contents

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.

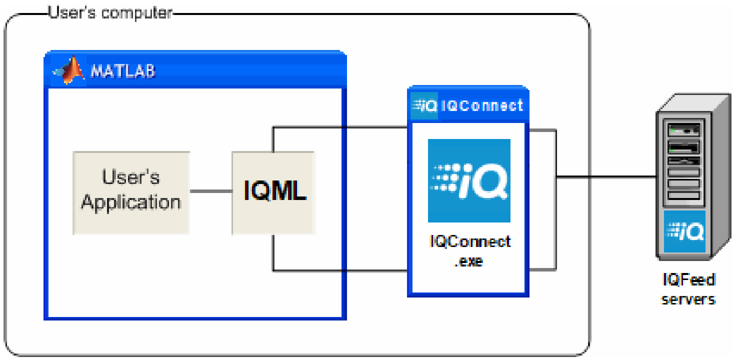
ALL TRADING SYMBOLS AND TRADING ORDERS DISPLAYED IN THE DOCUMENTATION ARE FOR ILLUSTRATIVE PURPOSES ONLY AND ARE NOT INTENDED TO PORTRAY A TRADING RECOMMENDATION.

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 *IQML* (www.IQML.net). *IQML* is a program that runs in Matlab and connects it to IQFeed. *IQML* 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.



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

1. A Java package (*IQML.jar*) that connects Matlab to IQFeed's client application.
2. A Matlab function (*IQML.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 (*IQML.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 (*IQML.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 *IQML*. Depending on the date that you installed *IQML*, your version may be missing some features discussed in this document. You can always download the latest *IQML* version from http://IQML.net/files/IQML.zip or https://UndocumentedMatlab.com/IQML/files/IQML.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 *IQML* 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 IQML

IQML 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
   On older Matlab releases, some *IQML* functionality may still be available. Contact info@IQML.net or info@UndocumentedMatlab.com for details.

Installing *IQML* is simple:

1. Read *IQML*'s license agreement.[2] This is required even for the trial version of *IQML*. If you do not accept the agreement you cannot use *IQML*.
2. Download *IQML.zip*[3] to a local folder (e.g.: *C:\IQML*).
3. Unzip the downloaded *IQML.zip* file in this local folder.
4. 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 *IQML*.
5. If you are running the Production (non-trial) version of *IQML*, you will need to activate your license at this point. When you purchase your license you will be given specific instructions for this. See §2.2 below for licensing details.
6. Ensure that the IQFeed client is working and can be used to log-in to IQFeed.[4]
7. You can now run *IQML* within Matlab. To verify that *IQML* is properly installed, retrieve the latest IQFeed server time, as follows (see §9.2 below):[5]
```
>> t = IQML('time');
```
8. You can query the installed version using *IQML***'s** 'version' action, as follows:
```
>> disp(IQML('version'))
    Version: 1.0
    Release: '23-Feb-2018'
    License: 'Professional'
     Expiry: '16-Jun-2018'
```
9. Once the *IQML* product is installed, you will be notified in the Matlab console (Command Window) whenever there is a new version available. You can always update your installation to the latest version, as follows:
```
>> IQML('update')
Downloading the latest IQML version from http://IQML.net/files/IQML.zip
 into C:\IQML\...
Download complete - installing...
Installation of the latest IQML version is now complete.
```

---

[2] http://IQML.net/files/IQML_License_Agreement.pdf or https://UndocumentedMatlab.com/IQML/files/IQML_License_Agreement.pdf

[3] http://IQML.net/files/IQML.zip or https://UndocumentedMatlab.com/IQML/files/IQML.zip

[4] *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 *IQML* 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.

[5] In some cases, you may need (or want) to specify the IQFeed **Username**,**Password** for the initial connection – see §9.1 below

2.2 Licensing and activation

IQML'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 *IQML* webserver. This activation key automatically validates online whenever *IQML* 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 *IQML*'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, *IQML* 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 *IQML*:

```
Error using IQML
IQML 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 *IQML* 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 *IQML* 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:

```
>> IQML('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 below).

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

```
*** Your IQML license will expire in 3 days (10-Mar-2018).
*** To extend your license please email info@IQML.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, *IQML* will stop working and display an error message:

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

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

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

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

Multiple *IQML* 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:

- **2- or 4-months** license: these short-term licenses can be repeatedly renewed, for product evaluation or program development beyond the free trial.
- **1-year** license: this is the standard, most popular license term.
- **Multi-year** license: 2-year, 3-year or 5-year license terms 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 *IQML* 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 *IQML* 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 *IQML* and *IB-Matlab* (the InteractiveBrokers-Matlab connector). The combination of IB+IQFeed+Matlab is quite common in trading systems.

Each of these licenses can be selected in one of two variants: **Standard** and **Pro**. The Standard license contains most IQFeed functionality; the Pro license provides access to the full set of IQFeed data. A detailed comparison is provided in §3.1, §3.4 below.

2.3 Switching activated computers

You can switch the *IQML* 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 *IQML'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.

2.4 Updating the installed version

Once *IQML* is installed, you will be notified in the Matlab console (Command Window) whenever a new version is available. An example of such a notification is:

```
>> IQML(...)  % some IQML command
A newer version of IQML (2.08) is available. Updates include:
  2.08 (2018-10-28)
    - Enabled parallelized historic data queries having date/time range
To display the latest online User Guide, click here.
To install the new version, click here, or run IQML('update'), or download
IQML.zip from http://IQML.net/files/IQML.zip and then unzip it in C:\IQML.
```

You can decide to ignore this notification and keep using your existing *IQML* version, or to follow the notification's advice and update your version – the choice is yours.

You can update *IQML* to the latest version any time during the license term, as follows:

```
>> IQML('update')
Downloading the latest IQML version from http://IQML.net/files/IQML.zip
 into C:\IQML\...
Download complete - installing...
Installation of the latest IQML version is now complete.
```

This update process preserves the current version as backup, so you could revert to it later (see below). Following the update, you can verify the new release's version:

```
>> data = IQML('version')
data =
    Version: 1.9
    Release: '16-Apr-2018'
    License: 'Professional'
     Expiry: '16-Jun-2018'
```

After installing the latest version, if you discover that this version does not work well, you can always revert back to a previous version:

1. run `IQML('revert')`, which replaces the current version with a previous version that was preserved in the last `IQML('update')`, or:
2. download http://IQML.net/files/IQML_previous.zip, unzip this file in your *IQML* folder, then restart Matlab. The current version is not preserved as backup, so you will not be able to revert to it later by running `IQML('revert')`.

After the version update (by either methods), restart Matlab, and run `data=IQML('version')` to verify the new version. Then email us to let us know why you reverted, so that we could correct the problem in upcoming versions.

3 Using IQML

3.1 General usage

IQML uses the IQFeed client[7] to connect to the IQFeed server. If an active IQFeed client is not detected, *IQML* 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 *IQML*. In any case, if an IQFeed connection is unsuccessful, *IQML* will error.

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

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

The *IQML* 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] = IQML(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 below.
- `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 = IQML('time');  %'time' action (blocking), 0 parameters
>> IQML('quotes', 'Symbol','IBM'); %streaming 'quotes' action, 1 parameter
>> IQML('command', 'String',command, 'PortName','Admin'); %2 parameters
```

Note that when an output `data` is requested, *IQML* 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 = IQML('time');  % blocking until data is available
```

When an output `data` is ***not*** requested, *IQML* 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 *IQML* in the background, and can later be retrieved using the mechanism that is discussed in §6 below. Examples of such non-blocking commands:

```
>> IQML('quotes', 'Symbol','IBM'); %start non-blocking IBM quotes stream
>> IQML('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 *IQML*, in the Standard and Professional licenses; trial licenses have the full functionality of a Professional license:

Action	**Description**	**User Guide Section(s)**	**Stan-dard**	**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 *IQML* installation to the latest version	§2.4	Yes	Yes
`'revert'`	Update the *IQML* 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 *IQML* from IQFeed	§9.1	Yes	Yes
`'reconnect'`	Disconnect and then re-connect *IQML* 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

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

- As name-value pairs – for example:
```
>> IQML('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';
>> IQML('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:
```
>> IQML('command', 'C:\MyData\inputFile.xlsx');
```
where:
  - Each column of the file contains a separate parameter
  - Row #1 contains the parameter names, and rows 2+ contain their corresponding values, one row per command
  - All commands must have the same `action` (`'command'` in this example)

For 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 *IQML* commands (by altering a few parameters), the entire set of parameters will be used, possibly including some leftover parameters from previous *IQML* commands. This may lead to unexpected results. For example:

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

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

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

In this example, the 2nd *IQML* command above will have no effect, because the **PortName** parameter in the `params` struct from the 1st *IQML* command will be reused in the 2nd command, sending it to the Level1 port, instead of to the Admin port. Similarly, the 3rd *IQML* 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 *IQML* command.

IQML 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:

```
>> IQML('quotes', 'Symbol','IBM');
>> IQML('quotes', 'symbol','IBM');
>> IQML('Quotes', "Symbol","IBM");
>> IQML('Quotes', 'Symbol','IBM');
>> IQML('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:

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

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

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

3.2 Common request properties

The following properties can be specified in *IQML*, with most `action`s:

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: • 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 *IQML*, the program will block until a response is available (i.e., a *synchronous* request). If no output parameter is specified, *IQML* 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 *IQML* in the background, and can later be retrieved using the mechanism that is discussed in §6 below. For example:

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

[8] In *IQML*, 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 *IQML* implies a default value of false (0) for **RaiseErrorMsgs** (see §3.5 below)

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:

```
>> IQML('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 = IQML('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:

```
IQML 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, `IQML(...,'Timeout','10')` rather than `IQML(...,'Timeout',10)`. Each *IQML* parameter expects a value of a specific data type, as listed in the parameter tables in this user guide. *IQML* 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 *IQML*'s functionality is available in all license types, some actions/functionality are only available in the Professional *IQML* license:

    - 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:[13]

```
>> data = IQML('news');
The 'news' action is not available in your Standard license of IQML, only
in the Professional license. Contact info@iqml.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@iqml.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 yyyymmdd 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#'
  LatestEventDatenum: 737128.637260451
LatestEventTimestamp: '20180309 15:17:39'
                 ...
```

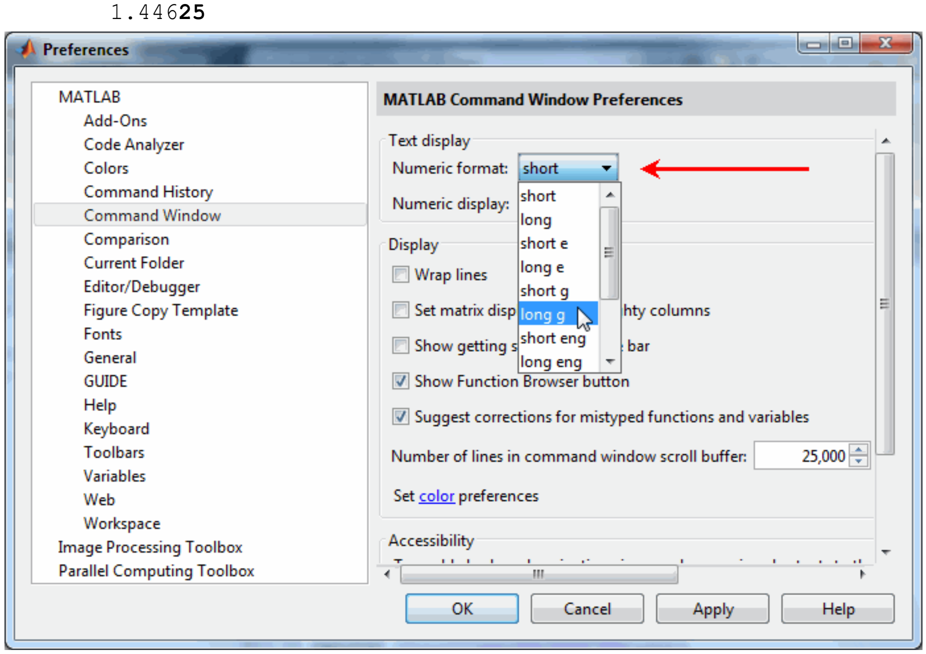
Depending on the data field, the timestamp is either your local computer's time, or IQFeed servers (New York) time – not the exchange time. To get the exchange time, you would need to do the appropriate time-zone arithmetic.

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.

IQML 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 = IQML('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 *IQML* return their data in the form of a struct-array (a Matlab array of structs), for example (see §8.6):

```
>> data = IQML('lookup', 'DataType','NAICS')
data =
  1175×1 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 = IQML('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.
```

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

```
>> [data, errorMsg] = IQML('quotes', 'Symbol','IBM', 'Timeout',0.1)
data =
     []
errorMsg =
    'IQML 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 *IQML* implies a default value of false (0) for the **RaiseErrorMsgs** parameter.

3.6 Run-time performance

3.6.1 General considerations

IQML'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 in quotes queries (§4.1, §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 msecs 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 *IQML* 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 msecs to process, the total processing time for the *IQML* query will be 20-40 secs.

When IQFeed is connected, it continuously sends messages to *IQML*: periodic "heartbeat" and status messages, and messages for any active streaming quotes or news events that you requested. These messages are automatically processed by *IQML* in the background, reducing the CPU time that is left available to process other *IQML* 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.

3.6.2 Paralellization

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

IQML 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 *IQML* 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`, *IQML* 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.

IQML 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 therefore best to start the parallel pool before time-critical operations, to avoid this startup time upon the first parallel query. Starting the pool 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 *IQML* 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.[15] Note that the parallel pool size should be set to ≤14, since IQFeed limits the number of concurrent connections.[16]
- In addition to the workers startup time, each worker independently connects to IQFeed upon the first *IQML* command it encounters, taking an extra few secs.
- 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).

---

[15] https://www.mathworks.com/help/distcomp/discover-clusters-and-use-cluster-profiles.html#f5-16540

[16] IQFeed's actual limit is 15, but one connection is used by the main (non-parallel) Matlab process, in addition to the workers.

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 = IQML('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 = IQML('news', 'DataType','headlines', 'MaxItems',100, ...
                   'GetStory',1, 'UseParallel',1); toc
Elapsed time is 15.799185 seconds.
```

3.6.3 Quote data-fields

Also in the Professional *IQML* license, you can customize the fields the IQFeed reports for market data quotes. 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 quotes queries, either for the latest snapshot (§4.1) or for streaming quotes (§6.1), will report just the requested fields. For example:

```
>> data = IQML('quotes', 'Symbol','AAPL', 'Fields',{'Bid','Ask','Last'})
>> data = IQML('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 *IQML*. By default, IQFeed reports 16 data fields.[17] However, ~50 additional fields can be requested (see §4.1 or §6.1 for details). Requesting fewer fields (as in the example above, which only requested 3 fields) will result in faster run-time processing. To improve run-time performance and reduce latency, request only those data fields that your program actually requires.

[17] 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

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 *IQML*'s 'quotes' action:

```
>> data = IQML('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 = IQML('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)

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. *IQML* 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).

```
>> IQML('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 = IQML('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 *IQML*’s timeout (default value: 5 secs) returns the results:

```
>> data = IQML('quotes', 'Symbol','IBM', 'NumOfEvents',4)
Warning: IQML 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 *IQML* 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 = IQML('quotes', 'Symbol','IBM', 'NumOfEvents',4, 'timeout',60);
```

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

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

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

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

```
data =
  1×3 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 *IQML* 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 = IQML('quotes', 'symbols',{'IBM','GOOG','AAPL'}, 'UseParallel',true);
```

Note that if you request quotes for a very large number of symbols in a single *IQML* 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 IQML: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 = IQML('quotes', 'fields',{})
data =
      CurrentFields: {1×16 cell}
    AvailableFields: {1×68 cell}

>> data.AvailableFields
ans =
  1×68 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) *IQML* 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 = IQML('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 *IQML*-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 *IQML* 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, *IQML* 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 valid for bonds. Similarly, EURUSD.FXCM's `Market_Capitalization` value is empty because Forex securities have no market cap. Likewise, `Net_Asset_Value` is only valid 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).

The **Fields** parameter can be set to any subset of `AvailableFields`,[22] 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 = IQML('quotes', 'Symbol','AAPL', 'Fields',{'Bid','Ask','Last'})
>> data = IQML('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 = IQML('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 = IQML('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 = IQML('quotes', 'Symbol','AAPL', 'Fields','Bid, Ask, xyz')
Error using IQML
Bad field 'xyz' was requested in IQML 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 *IQML*. To improve run-time performance and reduce latency, request only those data fields that are actually needed by your program.

[22] `AvailableFields` is reported by an `IQML('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** [23]	colon or comma-delimited string, or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: • '@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 if Parallel Computing Toolbox is installed, then querying multiple symbols will be done in parallel (see §3.6; Professional *IQML* license only).
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 *IQML* license only). The Symbol field is always returned first, even if not specified. Examples: • 'Bid:Ask:Last' • 'Bid, Ask, Last' • {'Bid', 'Ask', 'Last'} • 'All' (indicates all available fields)

The full list of available fields in IQFeed as of 1/7/2019 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 cen be retrieved in *IQML* using the `IQML('quotes','fields',{})` command, as explained above. For details about any of these fields, please contact your DTN/IQFeed representative (*IQML* just reports the data, it has no control over the reported values or definition of the data fields).

[23] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Field Name	Field Type	Description	Data origin [24]
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.[25]	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.[26]	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.	Exchange/other
Exchange ID	hexidecimal	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

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

[25] http://forums.iqfeed.net/index.cfm?page=topic&topicID=3891

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

Field Name	Field Type	Description	Data origin [24]
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 *IQML* 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 *IQML* 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	Possible single character values include: 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 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. A corresponding description field is generated by *IQML* for this field when **MsgParsingLevel** ≥ 2 (see §3.2)	IQConnect
Most Recent Trade	float	Price of most recent trade (inc. non-last-qualified trades).	Exchange/other
Most Recent Trade Aggressor	integer	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 []. Possible values: 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 (requires IQFeed client 6.1 or newer)	Exchange/other (only provided by some exchanges)
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 *IQML* for this field when **MsgParsingLevel** ≥ 2 (see §3.2, §8.7)	Exchange/other

Field Name	Field Type	Description	Data origin [24]
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 *IQML* 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 = IQML('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'
```

Notes:

1. the naming, interpretation and order of returned data fields is controlled by IQFeed, not by *IQML* – 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.[27] In any case, only the last 2 splits are reported by IQFeed.[28]
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 = IQML('fundamental', 'symbol','@ES#')
data =
  struct with fields:
                     Symbol: '@ES#'
                Exchange_ID: 22
                         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: 'Cantor Fitzgerald Exchange Treasury Funds'
  Listed_Market_Description: 'Chicago Mercantile Exchange Mini Sized
                              Contracts (CMEMINI)'
  Security_Type_Description: 'Future'
```

[27] 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. GOOG's 1:2.002 split on 3/27/2014) are not reported by IQFeed for some reason (probably a data error).

[28] 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 *IQML* to re-query IQFeed for fundamental data even when it is cached, set the **Debug** parameter:

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

It is possible to fetch fundamental data of multiple symbols in a single *IQML* command, by specifying a colon-delimited or cell-array list of symbols: [29]

```
>> data = IQML('fundamental', 'symbols','AAPL:GOOG')  %or: {'AAPL','GOOG'}
data =
  1×2 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. *IQML* 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 [30]
Symbol	string	The Symbol ID to match with watch request	
Exchange ID	hexidecimal	This is the Exchange Group ID (code). Convert to decimal and use the Listed Markets lookup to decode this value.	
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

[29] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

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

Field Name	Field Type	Description	Data origin [30]
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 of a security that were sold short by customers and securities firms that have not been repurchased to settle outstanding short market positions	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 two-digit month that the fiscal year ends for a company. For example, 4=April.	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
Format Code	string	Display format code	DTN
Precision	integer	Number of decimal digits	DTN

Field Name	Field Type	Description	Data origin [30]
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 [31]	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 quantitiy 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 [32] (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' [33]	IQML

[31] https://www.census.gov/eos/www/naics

[32] https://openfigi.com, https://omg.org/figi

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

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 = IQML('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).[34]

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 = IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',4)
data =
  4×1 struct array with fields:
    Symbol
    BarType
    ...
```

[34] 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

```
>> 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 *IQML*'s returned data as gaps in the `Timestamp`.

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

```
>> data = IQML('intervalbars', 'Symbol','IBM', 'NumOfEvents',4)
Warning: IQML 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
    BarType
    ...
```

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

```
>> data = IQML('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. *IQML* 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, *IQML* 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 *IQML* command, by specifying a colon-delimited or cell-array list of symbols. For example:

```
>> data = IQML('intervalbars', 'symbols',{'IBM','GOOG','AAPL'});
>> data = IQML('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 =
  100×3 struct array with fields:
    Symbol
    BarType
    Timestamp
    Open
    High
    Low
    Close
    CummlativeVolume
    IntervalVolume
    NumberOfTrades
```

However, if IQML 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 IQML returns empty results for this particular symbol:[35]

```
>> data = IQML('intervalbars', 'Symbol','IBM:GOOG:XXX', 'UseParallel',true)
data =
  1×3 cell array
    {100×1 struct}    {100×1 struct}    {0×0 double}
```

If you have the Professional license of *IQML* 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 = IQML('intervalbars', 'symbols',{'IBM','GOOG','AAPL'}, ...
               'UseParallel',1);
```

The following parameters affect interval bars data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [36]	colon or comma-delimited string or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: • '@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).

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

[36] In *IQML*, 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 *IQML* license only).
NumOfEvents	integer	Inf	One of: • 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: • '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, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'.
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.[37] For example:

```
>> data = IQML('marketdepth', 'symbol','@ES#')
data =
  10×1 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 struct display above. The `ID` field ('MD03' in this case) indicates 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 place, cancel or modify their orders.

Depending on the requested **Symbol**, IQFeed may report 5 to 15 market-depth rows.

[37] IQFeed does 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).

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 = IQML('marketdepth', 'symbol','@ES#', 'NumOfEvents',15)
data =
  10×1 struct array with fields:
    Symbol
    ID
    Bid
    Ask
    BidSize
    AskSize
    ...
```

The returned struct array can be converted into a Matlab table object, as follows:[38]

```
>> struct2table(data)
ans =
  10×14 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 = IQML('marketdepth', 'symbol','MSFT', 'NumOfEvents',50)
data =
  4×1 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.'
```

[38] 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 =
  4×14 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.[39] 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.[40] 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(IQML('marketdepth','symbol','MSFT','IncludeEmptyQuotes',1, ...
                     'NumOfEvents',70))
ans =
  67×14 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'
    ...
```

[39] 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: http://forums.iqfeed.net/index.cfm?page=topic&topicID=5594

[40] 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, *IQML* 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:[41]

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

The following parameters affect market depth data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [42]	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: • '@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 [43]	One of: • 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 *IQML* license.

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

[42] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

[43] The default value of **NumOfEvents** was changed from `inf` to 10 in *IQML* 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 = IQML('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_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
                           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: [1×1 struct]
         Underlying_Latest_Quote: [1×1 struct]
         This_Asset_Fundamentals: [1×1 struct]
         Underlying_Fundamentals: [1×1 struct]
```

The results are reported in a Matlab struct: the first few fields provide basic information on the derivative asset and its underlying security, followed by fair-value information, implied volatility and a long list of Greek values.

At the bottom of the returned data-struct, four sub-structs provide direct access to the latest quotes (§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.

The following Greek values are reported by *IQML*:

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	ν	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	$\partial \nu/\partial \sigma$	Sensitivity of Vega to changes in underlying asset's volatility; also sometimes called Volga
Veta	-	2	$\partial \nu/\partial \tau$	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 *IQML* command, by specifying the symbols using a colon-delimited string or a cell-array. For example:

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

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

```
data =
  2×1 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 = IQML('greeks', 'symbols',{'IBM1814L116','IBM1814X116'}, ...
               'UseParallel',true);
```

Notes:

1. Greeks and related derivative data (the the 'greeks' action in general) are only available in *IQML* Professional and trial licenses, not in the Standard license.
2. Greeks, fair-price and implied vol values are computed by *IQML* on your local computer. They are **NOT** provided by IQFeed, and are **NOT** approved by DTN.
3. There's 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.[44] Using *IQML*'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. *IQML* 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 *IQML*); *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 *IQML*, 365 by default);[45] 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.[46] In addition, some sources apparently have buggy math implementations.[47] The result is that different sources provide different Greek values for the same inputs. *IQML*'s values are basically identical to those of Matlab's Financial Toolbox, NAG and Maple.[48] Unfortunately, IQFeed's standalone Option Chains utility reports different values. *IQML* adheres to the core math formulae[49] and we believe that *IQML* provides mathematically-accurate results. However, the discrepancy between the values reported by different systems means that you must carefully ensure that *IQML*'s reported values fit your needs and expectations.

---

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

[45] Matlab's Financial Toolbox, NAG and Maple report annualized values; for annual values in *IQML*, set the **AnnualFactor** to 1.

[46] For example, the reported *Color* value is negative in NAG compared to *IQML* and Maple.

[47] This does not imply that there are no calculation bugs in *IQML*'s implementation; the Greeks calculation is not trivial.

[48] 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

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

By default, *IQML* 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). The `Underlying_Symbol` value can be overrriden 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 *IQML* 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 *IQML* 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 *IQML* by default. Instead of this reported value, you can specify another value (for example, the S&P 500 volatility), as a fixed percent value.

- **RiskFreeRate** – this is the domestic risk-free rate. *IQML* uses 0% by default; you can specify any other fixed percentage rate (based on e.g. LIBOR[50] or T-bill[51]).

- **DividendsYield** – this is the underlying asset's dividend yield. *IQML* 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, *IQML* determines the number of days until contract expiration (maturity) based on the contract's reported `Expiration_Date`. This maturity time can be overridden to any positive value. Note that the value is specified in days (not necessarily integer), not years.

- **AnnualFactor** – by default, *IQML* 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.

[50] You can query the current LIBOR rate with *IQML*, 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=mktindices), 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.

[51] You can query the current T-bill rate with *IQML*, 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 = IQML('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** [52]	colon-delimited string, or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: • '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).
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.
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 *Theta*, *Charm*, *Veta* and *Color* 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 *IQML* licenses, not in the Standard license.

[52] In *IQML*, 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 = IQML('summary')  % latest 5-minute data for all NYSE equities
data =
  4749×1 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 NYSE.[53] This data is daily (i.e., the cummulative 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).

[53] 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 = IQML('summary', 'DataType','fundamental')
data =
  4749×1 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 = IQML('summary', 'SecType','bond', 'Exchange','NYSE', ...
                          'DataType','fundamental');
```

```
>> struct2table(data)
ans =
  6326×11 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
'AA.28.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`.[54] 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 = IQML('summary', 'ReportEmptyFields',true)
data =
  4749×1 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);[55] 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).

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

[55] 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 = IQML('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 = IQML('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 = IQML('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** critera.

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 = IQML('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 IQML: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 *IQML*, 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 for 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.

IQML market summary query error: Code: 50004 - User not authorized for market summary file requested.

IQML market summary query error: Code: 50007 - No file available.
```

A related mechanism for fetching pre-filtered market scans for a select number of **Exchange**s and **Filter**s 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 symbols[56] with a few relevant data fields (far fewer fields than the snapshot queries), updated every 5 minutes during the trading day:

```
>> data = IQML('summary', 'DataType','top', 'Exchange','NYSE', 'Filter','active')
data =
  50×1 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 = IQML('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
```

[56] 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 = IQML('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.[57] Also note that the reported data may by 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:

<table>
<tr><th></th><th>Snapshot</th><th>Fundamental</th><th>Top</th></tr>
<tr><td>DataType</td><td>'snapshot'</td><td>'fundamental'</td><td>'top'</td></tr>
<tr><td>Exchange</td><td colspan="2">Multiple</td><td>Only AMEX, NYSE, NASDAQ</td></tr>
<tr><td>SecType</td><td colspan="2">Multiple</td><td>Only 'equity'</td></tr>
<tr><td>Date</td><td colspan="2">Latest (intra-day) or historic (end of day)</td><td>Only latest (updated every 5 minutes during the trading day)</td></tr>
<tr><td>Filter type</td><td colspan="2">Any Matlab function of any combination of data fields</td><td>Only one of 13 predefined types</td></tr>
<tr><td>Filter combinations</td><td colspan="2">Multiple criteria supported</td><td>Not supported</td></tr>
<tr><td>Processing time</td><td colspan="2">Tens/hundreds of secs</td><td><1 sec</td></tr>
<tr><td>IQFeed client</td><td colspan="2">Requires client 6.1 or newer</td><td>Any (client 5.0 or newer)</td></tr>
<tr><td>IQFeed connection</td><td colspan="2">Requires an active IQFeed connection (IQConnect login)</td><td>Does not require an active IQFeed connection</td></tr>
<tr><td>Result data fields</td><td>Up to 35 fields</td><td>Up to 47 fields</td><td>Only 11-13 data fields</td></tr>
<tr><td>Result records</td><td colspan="2">All securities that fit the parameters (many thousands)</td><td>Only the top 50 securities</td></tr>
</table>

[57] As of August 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: • 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: • '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: • '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 *IQML* 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').[58]

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 = IQML('history', 'symbol','IBM');
>> data = IQML('history', 'symbol','IBM', 'dataType','day') %equivalent
data =
  100×1 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:

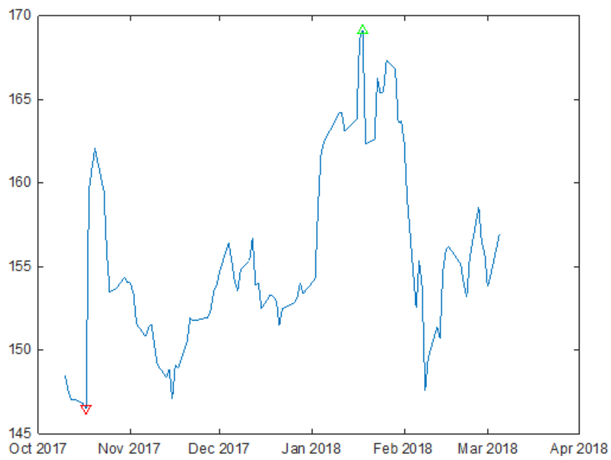
```
dates  = {data.Datestamp};      % cell-array of strings
closes = [data.Close];          % array of numeric values
```

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
```

58 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 = IQML('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 = IQML('history', 'symbol','@EMF19');
data =
  35×1 struct array with fields:
    Symbol
    ...
```

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

```
>> data = IQML('history', 'symbol','IBM', 'maxItems',20)
data =
  20×1 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.[59]

[59] 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, *IQML* might time-out on the request and return only partial data. Such a case is detected and reported by *IQML*:

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

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

```
>> data=IQML('history','symbol','IBM','maxItems',-1,'timeout',60) %oldest:1/2/96
data =
  5577×1 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 yyyymmdd (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**.[60]

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

```
>> data = IQML('history', 'symbol',{'IBM','GOOG','AAPL'}, 'maxItems',20)
>> data = IQML('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 =
  20×3 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 =
  1×3 cell array
    {77×1 struct}    {100×1 struct}    {55×1 struct}
```

[60] 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

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

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

The following parameters affect daily history data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [61]	colon or comma-delimited string or cell-array of strings	(none)	Limits query to specified symbol(s). Examples: • '@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: • 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)	Earliest bar date. Examples: • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' 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**.
EndDate	integer or string or `datetime`	'2099/12/31' (i.e., until now)	Latest bar date. See **BeginDate** parameter above for details.
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 *IQML* license only).

[61] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

5.2 Weekly data

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

```
>> data = IQML('history', 'symbol','FB', 'dataType','week')
data =
  100×1 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 = IQML('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 = IQML('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 *IQML* 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').[62]

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 = IQML('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/today[63] will be returned.

---

[62] 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. Also note that in some cases, depending on current day-of-week compared to the requested **BeginDate**, an addidional (older) bar might be returned that includes the week that was prior to the requested **BeginDate**.

[63] Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

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**.

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

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

>> data = IQML('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 =
  20×3 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 =
  1×3 cell array
    {77×1 struct}    {100×1 struct}    {55×1 struct}
```

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

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

The following parameters affect weekly history data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [64]	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: • '@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: • 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)	Earliest bar that includes a date. Examples: • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' Note: **MaxItems** has precedence over **BeginDate**: If there are more data points than **MaxItems** between **BeginDate** and yesterday/today[65], only the last **MaxItems** data points (from yesterday/today backward) will be returned, regardless of **BeginDate**.
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 *IQML* license only).

[64] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

[65] 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 = IQML('history', 'symbol','FB', 'dataType','month')
data =
  100×1 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 = IQML('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 = IQML('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 *IQML* 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 = IQML('history','symbol','FB','dataType','month','BeginDate',20180129);
```

Note: **<u>unlike</u>** daily data requests, you cannot specify **EndDate** in a request for historic monthly data bars. All monthly data bars up to yesterday/today[66] 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**.[67]

[66] Yesterday if using IQFeed client 6.0 or earlier; today if using IQFeed client 6.1 or newer

[67] 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 *IQML* command, by specifying a colon-delimited or cell-array list of symbols. For example:

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

>> data = IQML('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 =
  20×3 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 =
  1×3 cell array
    {77×1 struct}    {100×1 struct}    {55×1 struct}
```

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

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

The following parameters affect monthly 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: • '@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: • 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)	Earliest bar that includes a date. Examples: • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29' Note: **MaxItems** has precedence over **BeginDate**: If there are more data points than **MaxItems** between **BeginDate** and yesterday/today[69], only the last **MaxItems** data points (from yesterday/today backward) will be returned, regardless of **BeginDate**.
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 *IQML* license only).

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

[69] 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 = IQML('history', 'symbol','FB', 'dataType','interval')
data =
  100×1 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 time-zone), not `Datestamp` as with the other 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 =
  100×1 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=IQML('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 = IQML('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).[70]

By default, *IQML* 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, *IQML* only reports interval data from today. 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.

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.

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

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

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=IQML('history','symbol','FB','dataType','interval','dataDirection',-1);
```

[70] 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

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

```
>> data = IQML('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 *IQML* more time to gather data before returning the results.

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

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

>> data = IQML('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 =
  20×3 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 =
  1×3 cell array
    {77×1 struct}    {100×1 struct}    {55×1 struct}
```

IQML 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 *IQML* license and Matlab's Parallel Computing Toolbox (see §3.6):

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

>> data = IQML('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 *IQML* 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.[71]

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;[72] longer intervals (daily/weekly/monthly) can access up to 15+ years.[73]

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:

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

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

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.[74]

The following parameters affect interval history data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [75]	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: • '@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: • 1 means oldest-to-newest (default) • -1 means newest-to-oldest
LabelAtBeginning	integer	0	• 0: data at 11:17:41 is reported as '11:18' • 1: same data is reported as '11:17'

[71] 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)

[72] 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]*".

[73] Again, these values are for regular IQFeed accounts; IQFeed limits trial accounts (see note #71 above)

[74] http://forums.dtn.com/index.cfm?page=topic&topicID=5608

[75] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
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: • '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, hhmmss 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, hhmmss or hh:mm:ss
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). Only relevant when **Days**<0. Format: Matlab datenum, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'. 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) Only relevant when **Days**<0. Format: Matlab datenum, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'.
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 *IQML* license only).

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 = IQML('history', 'symbol','AAPL', 'dataType','ticks')
data =
  100×1 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.[76] Possible basis values are:[77]

- 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.[78] 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 `Datenum` 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 = IQML('history', 'symbol','@VX#', 'dataType','ticks');
>> data(1)
ans =
                 Symbol: '@VX#'
              Timestamp: '2019-10-04 09:42:41.499000'
                Datenum: 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 (…:57.899**000**). The actual time resolution of reported ticks depends on the specific exchange and security type.[79]

---

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

[77] Additional basis codes may be added by IQFeed in the future.

[78] http://forums.iqfeed.net/index.cfm?page=topic&topicID=3898

[79] 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.

You can limit the data that is returned, as with the historical-bars queries above:

By default, *IQML* only reports ticks data from today. 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.

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.

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

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

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).[80] 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.[81]

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

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

>> data = IQML('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
    ...
```

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

81 The above is true for IQFeed regular accounts; IQFeed trial accounts are limited to only 4 days of intraday data (see 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 =
  1×3 cell array
    {77×1 struct}    {100×1 struct}    {55×1 struct}
```

IQML 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 *IQML* license and Matlab's Parallel Computing Toolbox (see §3.6):

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

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

Finally, as with other *IQML* commands, you can set the **Timeout** parameter to a high value in order to allow *IQML* more time to gather data before returning the results.

The following parameters affect ticks history data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [82]	colon or comma-delimited string or cell-array of strings	(none)	Limits the query to the specified symbol(s). Examples: • '@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: • 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.

[82] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
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). Only relevant when **Days**<0. Format: Matlab datenum, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'. Note: **MaxItems** has precedence over **BeginDateTime**: If there are more data points than **MaxItems** between **BeginDateTime–EndDateTime**, only the last **MaxItems** data points (from **EndDateTime** backward) will be returned, regardless of **BeginDateTime**.
EndDateTime	integer or string or `datetime` object	'' (empty string) meaning now	Only return ticks that end before this date/time (US Eastern time-zone) Only relevant when **Days**<0. Format: Matlab datenum, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'.
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 *IQML* license only).

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 = IQML('summary', 'Date',20190110)  %all NYSE equities on Jan 10, 2019
data =
  4706×1 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':[83]

```
>> data = IQML('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.

[83] 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 phenomemon: 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.[84]

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 history for download (data is only available since May 20, 2018, and only for some combinations of **SecType/Exchange**). In all other cases, 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.
IQML market summary query error: Code: 50004 - User not authorized for market
summary file requested.
IQML market summary query error: Code: 50007 - No file available.
```

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: • 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: • '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 *IQML* license

[84] A [huge] static text file containing a [very long] list of expired symbols is available for download from DTN's FTP site (ftp://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 *IQML*.

6 Streaming data

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

These messages can either be queried asynchronously (via ad-hoc queries, as shown in §6.1-§6.4 below), or handled synchronously (using callbacks (§10) or alerts (§11)).

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. Later, whenever you wish to process the latest quote(s), simply use the 'quotes' action and **NumOfEvents** of -1 (minus one). This will return the latest information (a data struct), without stopping the background streaming.

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

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

IQFeed will start sending quotes to *IQML* 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 *IQML* command results to a variable, the request is treated as blocking and *IQML* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

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

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

The quotes are collected into an internal data buffer in *IQML*. 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**,[85] then up to the specified number of latest quotes will be stored. For example, to store the latest 5 quotes:

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

Note: **MaxItems** increases memory usage, multiplied by the number of streamed symbols.[86]

Subsequent requests to retrieve the latest accumulated quotes buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus one).

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

[86] Quotes use ~3KB of Matlab memory. So, if **MaxItems**=1500, then for 80 symbols *IQML* would need 80*1500*3KB = 360MB of Matlab memory when all 80 buffers become full (which could take a while).

These requests return a Matlab data struct similar to the following:

```
>> data = IQML('quotes', 'Symbol','@VX#', 'NumOfEvents',-1)
data =
                  Symbol: '@VX#'
                 Command: 'w@VX#'
                isActive: 0
         EventsToProcess: 10
         EventsProcessed: 10
      LatestEventDatenum: 737128.637260451
    LatestEventTimestamp: '20180309 15:17:39'
                DataType: 'quotes and trades'
             ProcessType: 'stream'
              BufferSize: 3
                  Buffer: [3×1 struct]
              LatestData: [1×1 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.
- `LatestEventDatenum` – Matlab numeric datenum representation of the `LatestEventTimestamp`.
- `LatestEventTimestamp` – local timestamp (string format) when this quote was received by *IQML*.
- `DataType` – type of data to stream (set by **DataType** parameter, see below).
- `ProcessType` – always equal to 'stream' for streaming quotes.
- `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.

Different quotes are sent independently from IQFeed server with a unique timestamp. Note: `data.LatestEventDatenum` 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:[87]

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

```
>> 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)'
```

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.[88]

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 *IQML* 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:[89]

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

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

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

IQFeed reports 16 standard data fields by default. If you have the Professional (or trial) *IQML* 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:

```
IQML('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.

88 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.

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

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 *IQML* command, by specifying a colon-delimited or cell-array list of symbols. For example:

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

And similarly, when retrieving the accumulated streaming data:

```
>> data = IQML('quotes', 'symbol','IBM:GOOG:AAPL', 'numOfEvents',-1);
data =
  1×3 struct array with fields:
    Symbol
    Command
    isActive
    EventsToProcess
    EventsProcessed
    LatestEventDatenum
    LatestEventTimestamp
    DataType
    ProcessType
    BufferSize
    Buffer
    LatestData
>> 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 *IQML* command. Note: this will return both active and non-active streams:

```
 >> data = IQML('quotes', 'numOfEvents',-1);  % no symbol: return ALL streams
data =
  1×5 struct array with fields:
    Symbol
    Command
    isActive
    ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to ''):[90]

```
>> IQML('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.[91]

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

[91] 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 *IQML* parameters that directly affect streaming quotes:

Parameter	Data type	Default	Description
Symbol or **Symbols** [92]	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: • '@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: • 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: • '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 *IQML* license only). The Symbol field is always returned first, even if not specified. Examples: • {'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.

[92] In *IQML*, 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. Later, whenever you wish to process the latest regional update(s), simply use the 'regional' action and **NumOfEvents** of -1 (minus one). This will return the latest information (a data struct), without stopping the background streaming.

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

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

This causes IQFeed to start sending regional updates to *IQML* 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 *IQML* command results to a variable, the request is treated as blocking and *IQML* will wait for all data to accumulate (or **Timeout** to occur), as described in §7.2:

```
IQML('regional', 'Symbol','FB', 'NumOfEvents',100);  % streaming, non-blocking
data = IQML('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 *IQML*. 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[93]. 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:

```
IQML('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.[94]

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

[94] 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 = IQML('regional', 'Symbol','FB', 'NumOfEvents',-1)
data =
                  Symbol: 'FB'
                 Command: 'S,REGON,FB'
                isActive: 0
         EventsToProcess: 100
         EventsProcessed: 100
      LatestEventDatenum: 737146.784037153
    LatestEventTimestamp: '20180327 18:49:00'
                DataType: 'regional'
             ProcessType: 'stream'
              BufferSize: 50
                  Buffer: [50×1 struct]
              LatestData: [1×1 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` >= `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.
- `LatestEventDatenum` – Matlab numeric datenum representation of the `LatestEventTimestamp`.
- `LatestEventTimestamp` – local timestamp (string format) when this update was received by *IQML*.
- `DataType` – always equal to 'regional' for streaming regional updates.
- `ProcessType` – always equal to 'stream' for streaming regional updates.
- `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:[95]

```
>> 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.LatestEventDatenum` 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**.[96]

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 *IQML* 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:[97]

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

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

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

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

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

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

[96] 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.

[97] During and around the time of the buffer clear, some streaming data may be lost, so it is advised not to clear to often…

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 = IQML('regional', 'numOfEvents',-1); % no symbol: get ALL streams
data =
  5×1 struct array with fields:
    Symbol
    Command
    isActive
    EventsToProcess
    ...
```

Similarly, to cancel all active streams in a single command, omit **Symbol** (or set it to ''):[98]

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

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

Parameter	Data type	Default	Description
Symbol or **Symbols** [99]	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: • '@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: • 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.

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

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

[99] In *IQML*, 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. Later, whenever you wish to process the latest interval bar(s), simply use the 'intervalbars' action and **NumOfEvents** of -1 (minus one). This will return the latest information (a data struct), without stopping the background streaming.

For example, request 600 streaming interval bars of a continuous VIX future contract:

```
IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',600)
```

This causes IQFeed to start sending interval bars to *IQML* 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 *IQML* command results to a variable, the request is treated as blocking and *IQML* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

```
IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',600);  % streaming, non-blocking
data = IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',600);  % 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 <u>not</u> enclose it in quotes (`'inf'`).

The quotes are collected into an internal data buffer in *IQML*. 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**,[100] then up to the specified number of latest quotes will be stored, subject to IQFeed server limitations:[101]

```
IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',600, 'MaxItems',3)
```

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

Subsequent requests to retrieve the latest accumulated interval bars buffer data, without stopping the background streaming, should use **NumOfEvents** = -1 (minus

---

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

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

[102] Interval bars use ~2KB of Matlab memory. So, if **MaxItems**=1500, then for 80 symbols *IQML* would need 80*1500*2KB = 240MB of Matlab memory when all 80 buffers become full (which could take a while).

one). These requests return a Matlab data struct similar to the following:

```
>> data = IQML('intervalbars', 'Symbol','@VX#', 'NumOfEvents',-1)
data =
                  Symbol: '@VX#'
                 Command: 'BW,@VX#,60,,1,3,,,B'
                isActive: 0
         EventsToProcess: 600
         EventsProcessed: 600
      LatestEventDatenum: 737128.637260451
    LatestEventTimestamp: '20180309 15:17:39'
                DataType: 'intervalbars'
             ProcessType: 'stream'
              BufferSize: 3
                  Buffer: [3×1 struct]
              LatestData: [1×1 struct]
        MaxDaysToProcess: 1
```

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.
- `LatestEventDatenum` – Matlab numeric datenum representation of the `LatestEventTimestamp`.
- `LatestEventTimestamp` – local timestamp (string format) when this bar was received by *IQML*.
- `DataType` – type of data to stream (set by **DataType** parameter, see below).
- `ProcessType` – always equal to 'stream' for streaming interval bars.
- `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.
- `MaxDaysToProcess` – maximal number of days with intervals data to process.

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: '2018-03-09 15:17:39'
                Open: 17.55
                High: 17.6
                 Low: 17.55
               Close: 17.6
    CummlativeVolume: 4550
      IntervalVolume: 11
      NumberOfTrades: 0
```

Note that `data.LatestData` is typically the same as `data.Buffer(end)`, regardless of the values of **MaxItems** or **NumOfEvents**.[103]

Different interval bars are sent independently from IQFeed server with a unique timestamp. Note that `data.LatestEventDatenum` and `data.LatestEventTimestamp` are specified in the local time-zone.

The `data.LatestData.BarType` field indicates whether this is a historic bar, or a bar from the live (real-time) stream, or an updated interval bar.

The `data.LatestData.NumberOfTrades` field indicates the number of trades that occurred within this bar (i.e., not cumulative), relevant only when **IntervalType** is '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).[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 *IQML* 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:[105]

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

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

[103] 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.

[104] 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

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

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

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

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

```
>> data = IQML('intervalbars', 'numOfEvents',-1); % no symbol: get ALL streams
data =
  5×1 struct array with fields:
    Symbol
    Command
    isActive
    ...
```

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

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

Interval bars can also fetch 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. *IQML* will return both the historical bars, as well as new 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.

The following parameters affect interval bars data queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [106]	colon or comma-delimited string, or cell-array of strings	(none)	Limits the request to the specified symbol(s). Examples: • '@VX#' • 'IBM:AAPL:GOOG' • 'IBM,AAPL,GOOG' • {'IBM', 'AAPL', 'GOOG'} This parameter must be set to valid symbol name(s) when **NumOfEvents**>0

[106] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

Parameter	Data type	Default	Description
NumOfEvents	integer	**MaxItems**	One of: • 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: • '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, 'yyyymmdd hhmmss' or 'yyyy-mm-dd hh:mm:ss'. Note: there is no corresponding **EndDateTime** parameter for streaming intervalbars (only for historic bars: §5.4).
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

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. Later, whenever you wish to process the latest market depth data, simply use the 'marketdepth' 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 market depth quotes for a continuous E-mini contract:

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

This causes IQFeed to start sending market depth updates to *IQML* in the background, up to the specified **NumOfEvents**, if defined, 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 *IQML* command results to a variable, the request is treated as blocking and *IQML* will wait for all the events to accumulate (or **Timeout** to occur), as described in §4.1:

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

NumOfEvents is an optional input parameter and can be any number higher than 1 for streaming to work. 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 *IQML*. 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 = IQML('marketdepth', 'Symbol','@ES#', 'NumOfEvents',-1)
data =
                  Symbol: '@ES#'
                 Command: 'w@ES#'
         EventsToProcess: 600
         EventsProcessed: 437
      IncludeEmptyQuotes: 0
              LatestData: [1×10 struct]
     LatestEventDatenum: 737195.518211377
   LatestEventTimestamp: '20180515 12:26:13'
```

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.
- `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.
- `LatestEventDatenum` – Matlab numeric datenum representation of the `LatestEventTimestamp`.
- `LatestEventTimestamp` – local timestamp (string format) when latest market depth quote was received by *IQML*.
- `ProcessType` – 'stream' to collect data in the background or 'block' to wait for data to come in and return it.

To retrieve the market depth data at the nth market depth row, simply read the fields of the `LatestData` at the nth 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.LatestEventDatenum` 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. Only the latest snapshot of the deep 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.

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

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

```
IQML('marketdepth', 'symbols',{'IBM','GOOG','AAPL'});
IQML('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 = IQML('marketdepth', 'Symbol','IBM', ...) %not subscribed to NYSE L2
Error using IQML
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 = IQML('marketdepth', 'numOfEvents',-1); % no symbol: get ALL streams
data =
  5×1 struct array with fields:
    Symbol
    Command
    isActive
    EventsToProcess
    ...
```

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

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

Refer to §4.4 above for a detailed description of the parameters that affect market depth data queries.

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

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 *IQML* license.

7.1 Configuration

To retrieve the news configuration for your account, set **DataType** to 'config':

```
>> data = IQML('news', 'DataType','config')
data =
    Category: 'All News'
      Majors: [1×7 struct]
>> {data.Majors.Source}
ans =
  1×7 cell array
    {'DTN'}    {'CPR'}    {'CBW'}    {'RTT'}    {'CPZ'}    {'CIW'}    {'BEN'}
>> {data.Majors.Description}
ans =
  1×7 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: [1×4 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: [1×0 struct]
>> data.Majors(7)
ans =
                Source: 'BEN'
           Description: 'Benzinga Pro'
    AuthenticationCode: '1a'
                IconID: 10
                Minors: [1×0 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 = IQML('news', 'DataType','headlines')
data =
  1000×1 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 *IQML* before the built-in **Timeout** (default: 5 secs) forces *IQML* to return the data to the user (remember, this is blocking mode, where a timeout applies):

```
>> data = IQML('news', 'DataType','headlines')
Warning: IQML 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 =
  738×1 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 *IQML* more time to gather the data before returning the results:

```
>> data = IQML('news', 'DataType','headlines', 'Timeout',10)
data =
  1000×1 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.[107] For example, to filter only headlines that relate to symbols BSX, BSX/AAPL, and BSX/AAPL/GOOG, respectively:

```
>> data = IQML('news', 'DataType','headlines', 'Symbols','BSX')
data =
  60×1 struct array with fields:
    Source
    ID
    ...

>> data = IQML('news', 'DataType','headlines', 'Symbols',{'BSX','AAPL'})
data =
  677×1 struct array with fields:
    Source
    ID
    ...

>> data = IQML('news', 'DataType','headlines', 'Symbols','BSX:AAPL:GOOG')
data =
  841×1 struct array with fields:
    Source
    ID
    ...
```

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 = IQML('news', 'DataType','headlines', 'Symbols','BSX:GOOG:AAPL', ...
                       'Sources','DTN:CPR:BEN')
data =
  745×1 struct array with fields:
    Source
    ID
    ...
```

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 above).

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 = IQML('news', 'DataType','headlines', 'Symbols','BZRatings');
```

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

You can limit the reported headlines to only a specific date, using the **Date** parameter:

```
>> data = IQML('news', 'DataType','headlines', 'Date',20180304, ...
                       'Symbols',{'BSX','AAPL'})
data =
  14×1 struct array with fields:
    Source
    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.[108]

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):

```
>> data = IQML('news', 'DataType','headlines', 'MaxItems',50)
data =
  50×1 struct array with fields:
    Source
    ID
    ...
```

By default, only the headline text is returned. To automatically fetch the full story text that is associated with each headline, set **GetStory** to `true`:

```
>> data = IQML('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
shareholders would receive between $3.00 and $3.50 per share in cash. ...
For comments and feedback: contact editorial@rttnews.com ↵Copyright(c) 2018
RTTNews.com All Rights Reserved'
```

Querying the story text for multiple headlines could take a long time. A rough estimate is that 2-3 full news stories can be retrieved sequentially each second. So for example, with 100 headlines, a news query with **GetStory**=`true` might take ~50 secs. If you have the Professional *IQML* license and Matlab's Parallel Computing Toolbox, you can parallelize this news query by setting **UseParallel** to `true`:

```
>> tic
>> data = IQML('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 = IQML('news', 'DataType','headlines', 'MaxItems',100, 'GetStory',1,...
               'UseParallel',1);
>> toc
Elapsed time is 15.799185 seconds.
```

108 https://help.dtniq.com/support-faqs

The following parameters affect (filter) news headlines queries:

Parameter	Data type	Default	Description
Symbol or **Symbols** [109]	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: • '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: • '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: • 737089 (Matlab datenum format) • datetime('Jan 29, 2018') • 20180129 (yyyymmdd format) • '20180129' • '2018/01/29' • '2018-01-29'
MaxItems	integer	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. This can be parallelized using the **UseParallel** parameter (see below).
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 stories for headlines using **GetStory**=`true` will be done in parallel (see §3.6; Professional *IQML* license only).

[109] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

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, §7.2 above). Different news sources provide their news stories in different formats, for example:

```
>> data = IQML('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 = IQML('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 = IQML('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 = IQML('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):

```
>> IQML('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 **ID**s in a single *IQML* query command, by specifying an array of values. For example:

```
>> data = IQML('news', 'DataType','story', 'ID',[22018991229,22018991585])

data =
  2×1 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 = IQML('news', 'DataType','number', 'Symbols','BSX')
data =
    BSX: 14
>> data = IQML('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 = IQML('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 above).

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 = IQML('news', 'DataType','number', 'Symbols','BSX:GOOG:AAPL',...
                       'BeginDate',20180301)
data =
    AAPL: 45
     BSX: 19
    GOOG: 15

>> data = IQML('news', 'DataType','number', 'Symbols','BSX:GOOG:AAPL',...
                       'BeginDate',20180301, 'EndDate',20180303)
data =
    AAPL: 37
     BSX: 3
    GOOG: 13

>> data = IQML('news', 'DataType','number', 'Symbols','BSX:GOOG:AAPL',...
                       'EndDate',20180305)
data =
    AAPL: 2038
     BSX: 191
    GOOG: 996

>> data = IQML('news', 'DataType','number', 'Symbols','BSX:GOOG:AAPL',...
                       'Date',20180301)
data =
    AAPL: 16
     BSX: 1
    GOOG: 3
```

IQML returns a Matlab struct, so the reported symbols need to be valid field names, and non-alphanumeric characters are automatically converted. For example:

```
>> data = IQML('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** [110]	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: • '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: • '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: • 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.

[110] In *IQML*, 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:

```
IQML('news', 'Symbols','FB:AAPL', 'NumOfEvents',100)
```

This causes IQFeed to start sending news headlines to *IQML* 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 *IQML* command results to a variable, the request is treated as blocking and *IQML* will wait for all the events to accumulate (or **Timeout** to occur), as described in §7.2:

```
IQML('news', 'NumOfEvents',100);         % streaming, non-blocking
data = IQML('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 <u>not</u> enclose it in quotes (`'inf'`).

The headlines are collected into an internal data buffer in *IQML*. 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`[111]. 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.[112]

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:

```
IQML('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.[113]

[111] 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 ('')

[112] 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.

[113] 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 = IQML('news', 'NumOfEvents',-1)
data =
                Command: 'S,NEWSON'
               isActive: 1
        EventsToProcess: 100
        EventsProcessed: 13
     LatestEventDatenum: 737146.726041343
   LatestEventTimestamp: '20180327 17:25:29'
               DataType: 'news'
            ProcessType: 'stream'
                Sources: {}
                Symbols: {}
             BufferSize: 50
                 Buffer: [13×1 struct]
             LatestData: [1×1 struct]
```

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

- `Command` – the command sent to IQFeed.[114]
- `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.
- `LatestEventDatenum` – Matlab numeric datenum representation of the `LatestEventTimestamp`.
- `LatestEventTimestamp` – local timestamp (string format) when this headline was received by *IQML*.
- `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.

[114] 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.LatestEventDatenum` 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**.[115]

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:

```
IQML('news', 'symbols',{'IBM','GOOG','AAPL'}, 'numOfEvents',6);
IQML('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:

```
IQML('news', 'Symbols','BZRatings:FB:AAPL', 'numOfEvents',6);
```

Note: if you omit the **Symbols** parameter in your *IQML* 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:

```
IQML('news', 'sources',{'DTN','CPR','BEN'}, 'numOfEvents',6);
IQML('news', 'sources','DTN:CPR:BEN', 'numOfEvents',6);  % equivalent
```

[115] 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 *IQML* command, no filtering of headlines based on their source will be performed, and all headlines will be collected.

Here is a summary of the *IQML* parameters that affect streaming news headlines:

Parameter	Data type	Default	Description
Symbol or **Symbols** [116]	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: • '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: • 'DTN' • 'DTN:CPR:BEN' • 'DTN,CPR,BEN' • {'DTN', 'CPR', 'BEN'}
NumOfEvents	integer	Inf	One of: • 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.

[116] In *IQML*, 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 = IQML('lookup', 'DataType','symbols', 'Name','IBM')
data =
  1086×1 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 = IQML('lookup', 'DataType','symbols', 'name','ibm', ...
                         'Description','business')
data =
  8×1 struct array with fields:
    Symbol
    Description
    Market_ID
    Market_Name
    Sec_Type_ID
    Sec_Type

>> data = struct2table(data)
data =
  8×6 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:[117]

```
>> data = IQML('lookup', 'DataType','symbols', 'Description','inc', 'SIC',83)
data =
  6×1 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'
```

[117] 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).[118]

Note that it is possible that not all the requested symbols will be received before *IQML*'s timeout (default value: 5 secs) returns the results:[119]

```
>> data = IQML('lookup', 'DataType','symbols', 'Name','GOOG')
Warning: IQML timeout: only partial data is returned. Perhaps the Timeout
parameter should be set to a value larger than 5
data =
  3848×1 struct array with fields:
    Symbol
    Description
    Market_ID
    Market_Name
    Sec_Type_ID
    Sec_Type
```

To control the maximal duration that *IQML* 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 = IQML('lookup', 'DataType','symbols', 'Name','GOOG', 'timeout',30)
data =
  6812×1 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 = IQML('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 = IQML('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**, **SIC**, **NAICS**:

```
>> data = IQML('lookup', 'DataType','symbols')
Either Name, Description, SIC or NAICS parameters must be specified in a
symbol lookup query
```

[118] 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 below for details

[119] *IQML* 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(IQML('lookup', 'datatype','symbols', 'name','GOOG', 'SecType','Equity'))
ans =
  2×6 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 = IQML('lookup', 'datatype','symbols', 'name','GOOG', 'Market','NGSM');
```

Multiple **Markets** and/or **SecTypes**[120] can be specified using a cell array. For example, to get the list of all active (non-expired) GOOG equities and options:[121]

```
>> data = IQML('lookup', 'datatype','symbols', 'name','GOOG', ...
               'SecTypes',{'Equity','IEOption'}, 'Timeout',20)
data =
  8056×1 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 = IQML('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 = IQML('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 = IQML('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,[122] as a scalar integer value or as a numeric array of integers:

```
>> data = IQML('lookup','datatype','symbols','name','GOOG','SecTypes',1);
>> data = IQML('lookup','datatype','symbols','name','GOOG','SecTypes',[1,2]);

>> data = IQML('lookup','datatype','symbols','name','GOOG','Markets',21);
>> data = IQML('lookup','datatype','symbols','name','GOOG','Markets',[21,14]);
```

[120] Note that you can use either **Market** or **Markets** as the parameter name, and similarly, either **SecType** or **SecTypes**.

[121] IQFeed only returns the symbols of active (non-expired) options/futures. See §8.2 below for details about expired contracts.

[122] 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 *IQML* 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, *anywhere* within the symbol name)
Description	string	'' (empty string)	Limits the query to assets that contain the specified string in their description (case insensitive, *anywhere* within the description)
Market or **Markets** [123]	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** [124]	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 *beginning* 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 *beginning* 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)

[123] In *IQML*, the **Market** and **Markets** parameters are synonymous – you can use either of them, in any capitalization

[124] In *IQML*, 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:[125]

```
>> symbols = IQML('chain', 'Symbol','GOOG')'  % options chain for GOOG
symbols =
  1×1454 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'
  ...
```

All chain queries support the **Symbol**, **Months**, **Years**, and **NearMonths** parameters (filtering criteria – see table below). The options-related chain queries (**DataType**= 'options' or 'foptions') also support a **Side** parameter ('cp' (default), 'c' or 'p' – to limit the reported options to calls and/or puts). In addition, the index/equity options chain query (**DataType**='options') also supports **IncludeBinary**, **MinStrike/MaxStrike** and **NumInMoney/NumOutOfMoney** filtering parameters. For example:

```
% Report GOOG options having strike price between $1000-$1010 in next 4 months
>> symbols = IQML('chain', 'symbol','goog', 'NearMonths',4, ...
                           'MinStrike',1000, 'MaxStrike',1010)'
symbols =
  1×58 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'
  ...
```

Note that if you filter by **MinStrike** and/or **MaxStrike**, you cannot also filter by **NumInMoney/ NumOutOfMoney** (and vice versa):

```
>> IQML('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:

```
>> IQML('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 *IQML* query will return an empty cell array:

```
>> symbols = IQML('chain', 'datatype','spreads','symbol','C','years',2010:2019)
symbols =
  0×0 empty cell array
```

[125] 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: IQFeed only returns active (non-expired) contracts. A [huge] static text file containing a [very long] list of expired symbols is available for download,[126] and starting with IQFeed client 6.1 you can also fetch this data using *IQML* (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 = IQML('chain', 'symbol','GOOG', 'NearMonths',4, ...
                        'MinStrike',1000, 'MaxStrike',1010, ...
                        'WhatToShow','quotes')
data =
  58×1 struct array with fields:
    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
    ...

>> 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'
    Most_Recent_Trade_Market_Center: 156
                     Total_Volume: 0
                              Bid: 140.5
                         Bid_Size: 3
                              Ask: 150.1
                         Ask_Size: 1
                             Open: []
                             High: []
                              Low: []
                            Close: 120
                 Message_Contents: 'Cbacv'
              Message_Description: 'Last qualified trade; A bid update occurred;
An ask update occurred; A close declaration occurred; A volume update occurred'
       Most_Recent_Trade_Conditions: 1
       Trade_Conditions_Description: 'Normal Trade'
            Most_Recent_Market_Name: 'MIAX PEARL Options exchange'

>> symbols = {data.Symbol}
symbols =
  1×58 cell array
  Columns 1 through 4
    'GOOG1803H1000'    'GOOG1803H1010'    'GOOG1810H1000'    'GOOG1810H1005'
  Columns 5 through 8
    'GOOG1810H1010'    'GOOG1813G1000'    'GOOG1813G1002.5'  'GOOG1813G1005'
  ...
```

[126] ftp://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 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 =
  1×58 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 = IQML('chain', 'symbol','GOOG', 'NearMonths',4, ...
                        'MinStrike',1000, 'MaxStrike',1010, ...
                        'WhatToShow','fundamental')
data =
  58×1 struct array with fields:
    Symbol
    Exchange_ID
    PE
    Average_Volume
    x52_Week_High
    x52_Week_Low
    Calendar_Year_High
    Calendar_Year_Low
    ...

>> 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
  Columns 41 through 48
     1005      1010      1000    1002.5      1005    1007.5      1010      1000
  Columns 49 through 56
     1005      1010      1000      1005      1010      1000      1005      1010
  Columns 57 through 58
     1000      1010
```

Here is a summary of the *IQML* 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: • 'options' (default) – on index/equity • 'future' • 'spread' – future calendar spreads • 'foptions' – options on future
Side	string	'cp' (meaning both calls and puts)	One of: • 'cp' (default) – both calls and puts • 'c' – calls only • 'p' – puts only Only relevant if **DataType**='options'/'foptions'
WhatToShow	string	'symbols'	One of: • 'symbols' (default) – list of symbols in chain • 'quotes' – return the latest quotes data • 'fundamental' – return fundamental data
Months	various	[] meaning all	One of: • 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 FGHJKMNQUVXZ (e.g. 'JKMN') Cannot be specified together with **NearMonths**
NearMonths	integer (0-99)	[]	Number of nearby contract months to report.[127] Cannot be specified together with **Months**.
Years	integer scalar/array	[] meaning current year	One or more years (e.g. 2013:2019). Default = current year.
IncludeBinary	logical	false or 0	If true, binary/weekly options[128] 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'.
NumOutOf Money	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).

Note: Options/futures chain lookup is only available in the Professional *IQML* license.

[127] 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

[128] 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 = IQML('lookup', 'DataType','markets')
data =
  474×1 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 =
  9×5 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 =
  9×5 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 the **Name** and/or the **Description** filtering parameters. For example, let's display only the markets that contain 'Nasdaq' in their **Description**:

```
>> data = IQML('lookup', 'DataType','markets', 'Description','Nasdaq')
data =
  10×1 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 = IQML('lookup', 'DataType','markets', 'Name','xyz')
data =
     []

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

Here is a summary of the *IQML* parameters that affect markets lookup:

Parameter	Data type	Default	Description
Name	string	'' (empty string)	Limits the query to markets that contain the specified string in their name or groupName (case insensitive, *anywhere* within the name)
Description	string	'' (empty string)	Limits the query to markets that contain the specified string in their description (case insensitive, *anywhere* within the description)

8.4 Security types lookup

To retrieve a list of security types, set action to 'lookup' and **DataType** to 'sectypes':

```
>> data = IQML('lookup', 'DataType','sectypes')
data =
  38×1 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 the **Name** and/or the **Description** filtering parameters. For example, let's display only security types that have 'Option' in their **Description**:

```
>> struct2table(IQML('lookup', 'DataType','sectypes', 'Description','option'))
ans =
  4×3 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 = IQML('lookup', 'DataType','sectypes', 'Name','xyz')
data =
     []

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

Here is a summary of the *IQML* parameters that affect security types lookup:

Parameter	Data type	Default	Description
Name	string	'' (empty string)	Limits the query to secTypes that contain the specified string in their name (case insensitive, *anywhere* within the name)
Description	string	'' (empty string)	Limits the query to secTypes that contain the specified string in their description (case insensitive, *anywhere* within the description)

8.5 SIC codes lookup

To retrieve a list of SIC sectors/industries, set action to 'lookup' and **DataType** to 'SIC':

```
>> data = IQML('lookup', 'DataType','SIC')
data =
  1009×1 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 the **Description** filtering parameter. For example, let's display only security types that have 'Oil' in their **Description**:

```
>> struct2table(IQML('lookup', 'DataType','SIC', 'Description','oil'))
ans =
  22×2 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 = IQML('lookup', 'DataType','SIC', 'Description','xyz')
data =
     []

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

Here is a summary of the *IQML* parameters that affect SIC codes lookup:

Parameter	Data type	Default	Description
Description	string	'' (empty string)	Limits the query to SIC entries that contain the specified string in their description (case insensitive, *anywhere* 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).[129] 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).

[129] 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 = IQML('lookup', 'DataType','NAICS')
data =
  1175×1 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 **Description** filtering parameter. For example, let's display only security types that have 'Oil' in their **Description**:

```
>> struct2table(IQML('lookup', 'DataType','NAICS', 'Description','oil'))
ans =
  20×2 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' [130]
   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 = IQML('lookup', 'DataType','NAICS', 'Description','xyz')
data =
     []

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

Here is a summary of the *IQML* parameters that affect NAICS codes lookup:

Parameter	Data type	Default	Description
Description	string	'' (empty string)	Limits the query to NAICS entries that contain the specified string in their description (case insensitive, *anywhere* 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).[131] It is possible that a similar bug also applies to NAICS.

[130] The extra space at the beginning of the description here is a typo in IQFeed's data

[131] 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 = IQML('lookup', 'DataType','conditions')
data =
  155×1 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 the **Name** and/or the **Description** filtering parameters. For example, let's display only security types that have 'Option' in their **Description**:

```
>> struct2table(IQML('lookup', 'DataType','conditions', 'Description','option'))
ans =
  7×3 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 = IQML('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.[132] For example (see §4.1):

```
>> data = IQML('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')[133] and hexadecimal 87 (=135, meaning 'Odd lot trade').

Here is a summary of the *IQML* parameters that affect trade conditions lookup:

Parameter	Data type	Default	Description
Name	string	'' (empty string)	Limits the query to trade conditions that contain the specified string in their name (case insensitive, *anywhere* within the name)
Description	string	'' (empty string)	Limits the query to trade conditions that contain the specified string in their description (case insensitive, *anywhere* in the description)

[132] 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).

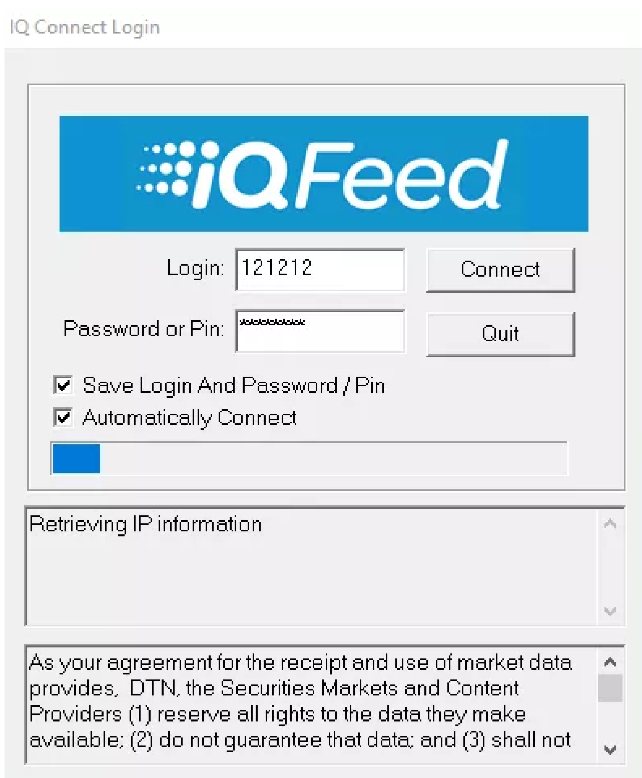
[133] The missing "r" in "Intramarket" is a typo in IQFeed's data

9 Connection, administration and other special commands

9.1 Connecting & disconnecting from IQFeed

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

IQML 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):

```
>> IQML('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 *IQML* 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 below).

If you enter an invalid set of **Username**/**Password**, an error message will be thrown. A different error will be thrown if *IQML* fails to connect to IQFeed within 10 seconds.

IQML 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 *IQML*), even without **Username** and **Password** in the initial *IQML* connection. *IQML* will bypass login, connecting directly to the client process.

IQML typically uses the latest API features supported by your installed IQFeed client (*IQConnect*). For example, if you use client version 6.0.1.1, *IQML* will use IQFeed communication protocol 6.0, and you will not have access to features of protocol 6.1.[134] Once you install a newer IQFeed client, *IQML* will automatically detect this and use a newer protocol number, 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, but not to 6.1 or newer:

```
>> IQML(..., 'Protocol',6.2)
Warning: Your IQFeed client (version 6.0.1.1) does not support Protocol 6.1 -
using Protocol 6.0 instead
(Type "warning off IQML:UnsupportedProtocol" to suppress this warning.)
```

You will be able to retrieve information in Matlab as soon as *IQML* 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 *IQConnect*
Password	string	(none)	Used to launch and login into *IQConnect*
Protocol	number	installed client's version	IQFeed API protocol that should be used. You can set any protocol value supported by your *IQConnect* 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, *IQML* uses the following TCP ports to connect to IQFeed's client:

- 5009 – Level1Port – used for Level 1 queries/msgs (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 *IQML* via these ports. *IQML* automatically tries to detect and report such cases, for example:

```
>> data = IQML('history', 'Symbol','IBM')
Error using IQML
Port #9100 is already used by process #12345 (SomeProcess.exe) and cannot be
used by IQFeed. Run IQML('registry') to set a different value for LookupPort.
```

You can solve such port conflicts using any of the following methods:

[134] For example, historic market summary data and scanning (§5.6)

1. Stop/uninstall the other process (**SomeProcess.exe** in the example above)
2. Modify the other process's settings to use different ports, so that they will not conflict with IQFeed (this is rarely possible)
3. Modify IQFeed's registry settings to use a different port, which is not used by any other process (see §9.5 below). 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 IQML command. An *IQConnect* restart is necessary since *IQConnect* reads the registry values and sets the ports only when it starts.

In some cases, users may wish to disrupt a live connection. You can disconnect from IQFeed using *IQML*'s 'disconnect' action, which has no settable parameters:

```
>> IQML('disconnect')
```

This command disconnects *IQML* from the IQFeed client. If *IQML* 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.

There is no need for a corresponding **connect** action, because connection is automatically (re-)established whenever this is required by a new *IQML* command.

IQML 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
```

In some cases, users may wish to actively re-connect (disconnect and then connect) to IQFeed. This can be done with the 'reconnect' action (no settable parameters):

```
>> IQML('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 = IQML('time')
data =
       latestEventDatenum: 737114.660205451
     latestEventTimeStamp: '20180223 15:50:41'
      latestServerDatenum: 737114.368518519
    latestServerTimestamp: '20180223 08:50:40'
```

The returned data struct includes the following data fields:

- `latestEventDatenum` – 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 `latestEventDatenum`
- `latestServerDatenum` – a Matlab numeric datenum value that corresponds to the latest **server** time that was received from IQFeed.
- `latestServerTimeStamp` – a human-readable format of `latestServerDatenum`

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 `latestServerDatenum` lags by 0.5 secs behind the current time on average.

Similarly, `latestEventDatenum` 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 `latestServerDatenum`.

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 = IQML('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: {1×16 cell}
                  ServerVersion: '6.0.0.5'
                    ServiceType: 'real_time'
```

The returned data struct includes the following data fields:[135]

- `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` – The # of symbols that are currently being streamed
- `NumOfClientsConnected` – The # of clients that are currently connected
- `SecondsSinceLastUpdate` – The # of seconds since the last update from the Quote Server
- `NumOfReconnections` – The # of times that IQFeed successfully reconnected
- `NumOfAttemptedReconnections` – The # of times that IQFeed has attempted to reconnect, but failed
- `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 *IQML* (i.e., *IQML* commands/requests to IQFeed). Found in the "Internet Bandwidth" section of the IQConnection Manager. Formula: total bytes received / 1024

[135] http://iqfeed.net/dev/api/docs/AdminSystemMessages.cfm

- `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
- `TotalKBsSent` – Total # of Kilobytes sent from IQFeed to *IQML* (i.e., IQFeed messages to *IQML*). 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` – The list of exchanges for which this IQFeed account is subscribed
- `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 = IQML('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: {1×16 cell}
                  ServerVersion: '6.0.0.5'
                    ServiceType: 'real_time'
                         Level2: [1×1 struct]
           Level2SymbolsWatched: 2
                         Lookup: [1×1 struct]
          RegionalSymbolsWatched: 2
                          Admin: [1×1 struct]
                         Level1: [1×1 struct]
           Level1SymbolsWatched: 0
```

```
>> data.Level1
ans =
    ConnectTime: '20180401 202111'
    KBsReceived: 0.74
        KBsSent: 70.58
      KBsQueued: 0

>> data.Admin
ans =
    ConnectTime: '20180401 202108'
    KBsReceived: 0.43
        KBsSent: 1516.74
      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):[136]

- `ConnectTime` – Timestamp when this channel to IQFeed was first opened
- `KBsReceived` – Total # of Kilobytes sent from *IQML* to IQFeed via this channel (requests for data queries etc.).
- `KBsSent` – Total # of Kilobytes sent from IQFeed to *IQML* 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 *IQML* 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.

[136] 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,TIMESTAMPSOFF' command,[137] which stops IQFeed from sending server timestamp messages every second:

```
>> IQML('command', 'String','S,TIMESTAMPSOFF')
```

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,TIMESTAMPSOFF' command is defined for the Level1 port,[138] whereas the 'S,CLIENTSTATS OFF' command (which stops the IQFeed server from streaming client stats messages) is defined for the Admin port.[139] When you use *IQML*'s standard actions you do not need to worry about which port handles which command – this is automatically handled by *IQML*. 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:

```
>> IQML('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 are not entirely consistent in the format of the various commands. For example, the 'S,TIMESTAMPSOFF' command has no space, whereas the 'S,CLIENTSTATS OFF' command does have a space; also, both of these commands are all-uppercase, yet 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 *IQML* 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: • 'Level1' (default) • 'Level2' • 'Lookup' • 'Admin'
ProcessFunc	function handle	[]	Custom user callback function to process incoming IQFeed data messages (see §10).

137 http://iqfeed.net/dev/api/docs/Level1viaTCPIP.cfm

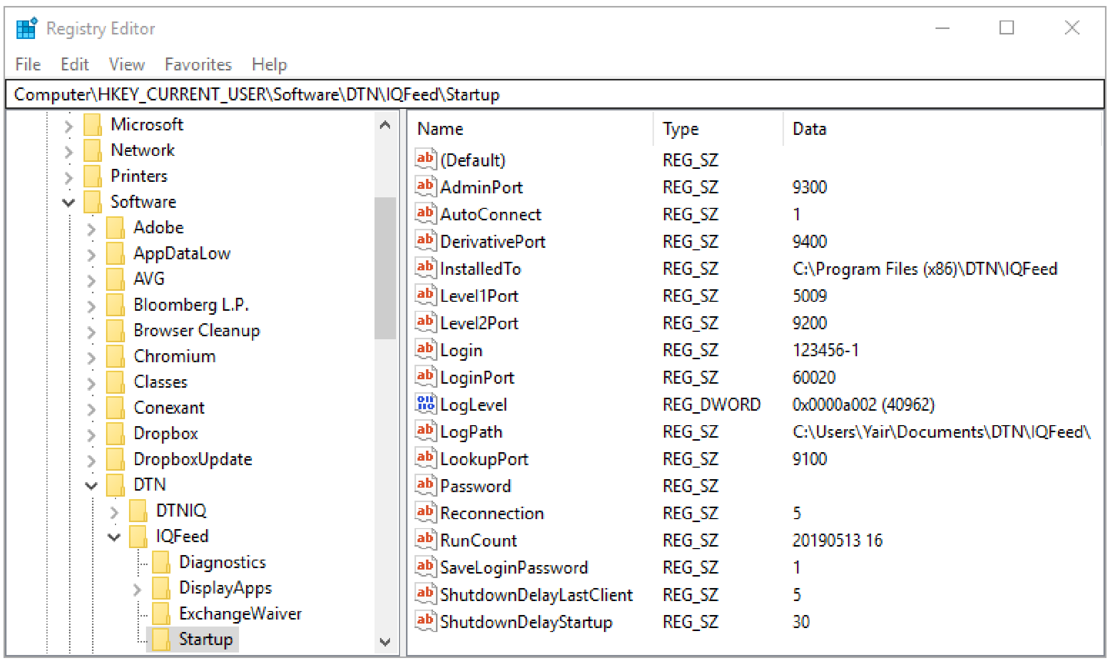
138 http://iqfeed.net/dev/api/docs/Level1viaTCPIP.cfm

139 http://iqfeed.net/dev/api/docs/AdminviaTCPIP.cfm

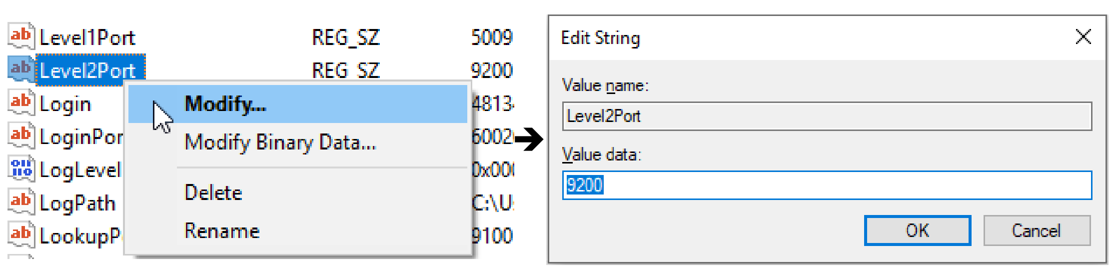
9.5 Modifying IQFeed's registry settings

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 *IQML* 'registry' action, which has no settable parameters:[140]

```
>> IQML('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.

---

[140] Note: running Windows Registry Editor requires local computer Administrator priviledges (or elevation).
In Mac/Linux, IQML('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 IQML

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.[141] Whereas Matlab callbacks are normally used in conjunction with Graphical User Interfaces (GUI), they are also used with *IQML*, 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 *IQML* 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 *IQML* is called **ProcessFunc**.

There are two types of callbacks that you can use in *IQML*:

- *Generic callback* – this is a catch-all callback function that is triggered upon <u>any</u> 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 *IQML*, the command/action does not need to be related to the callback. For example:

```
data = IQML('time', 'ProcessFunc',@IQML_Callback);
```

where `IQML_Callback()` is a Matlab function created by you that accepts two input arguments, which are automatically populated in run-time:

[141] http://www.mathworks.com/help/matlab/creating_guis/writing-code-for-callbacks.html

- `iqObject` – this is currently an empty array. Future versions of *IQML* 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 IQML_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.[142] For example, to pass two extra data values:

```
callbackDefinition = {@IQML_Callback, 123, 'abc'};
IQML('time', 'ProcessFunc',callbackDefinition);

function IQML_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). IQFeed messages always begin with a single character indicating the message type:

```
    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: {1×20 cell}
```

All IQFeed messages typically begin with a single character followed by ',', which we call the `MessageHeader`, and which identify the `MessageType`. For example, `MessageHeader` of 'T' identifies a Time message, and 'S' identifies a System message.[143]

[142] http://www.mathworks.com/help/matlab/creating_guis/writing-code-for-callbacks.html#brqow8p

[143] 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);

>> IQML('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 = @IQML_Quote_Update_Callback;
>> IQML('time','ProcessFunc',callbacks);
```

Here is a more elaborate example, were 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',@IQML_Quote_Update_Callback, ...
                      'Default',@IQML_Default_Callback);

% Alternative #2: using direct field assignments:
>> callbacks.Time = 'disp TIME!';
>> callbacks.System = [];  % ignore System messages
>> callbacks.Quote_update = @IQML_Quote_Update_Callback;
>> callbacks.Default = @IQML_Default_Callback);

>> IQML('time','processFunc',callbacks);
```

When you specify any callback function to *IQML*, you only need to set it once, in any *IQML* command. Unlike most *IQML* parameters, which are not remembered across *IQML* commands and need to be re-specified, callbacks do <u>not</u> 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 *IQML* command, it is simply useless and makes the code less readable.

To reset all callbacks (i.e., remove any callback invocation), simply set the **ProcessFunc** parameter value to [] (empty square brackets):

```
IQML('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 = @IQML_Default_Callback);

>> IQML('time','ProcessFunc',callbacks);
```

Matlab callbacks are invoked even if you issue a custom IQFeed command (see §9.4). This is actually very useful: you can use the command to 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 or even a single character. 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 *IQML* completes in the fastest possible time. This is important for programming in general, but it is especially important for *IQML* callbacks, which are invoked (executed) every time that a new message arrives from IQFeed, numerous times each second.

As explained in §3.6, *IQML*'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, *IQML* can process up to 500-1000 IQFeed messages per second.

When you add your own user-defined callbacks, their processing time is added to *IQML*'s. For example, if your callback takes an average of just 3 msecs to process (which is quite fast), then the total average message processing time will be 4-5 msecs, lowering *IQML*'s effective maximal processing rate from 500-1000 to just 200-250 messages/second. The more callbacks and alerts that you define, and the longer each of them takes to process, the lower will *IQML*'s message processing rate 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 a disk, 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 tool[144] to check your callback code for run-time performance hotspots that can be optimized to run faster.
10. Read the textbook "*Accelerating MATLAB Performance*",[145] authored by *IQML*'s creator (see §15.2), for numerous tips on improving Matlab run-time.

---

[144] https://mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html

[145] https://undocumentedmatlab.com/books/matlab-performance

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 *IQML*'s custom command functionality (§9.4), specifying a custom callback function for the 'Chain' MessageType:[146]

```
% Equity options chain for GOOG:
processFunc.Chain = @IQML_Chain_Callback;
>> IQML('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,GOOG1806P1
005,GOOG1806P1007.5,GOOG1806P1010,GOOG1806P1012.5,GOOG1806P1015,GOOG1806P1017
.5,GOOG1806P1020,GOOG1806P1022.5,GOOG1806P1025,GOOG1806P1027.5,GOOG1806P1030,
GOOG1806P1032.5,GOOG1806P1035,GOOG1806P1037.5,GOOG1806P1040,GOOG1806P1042.5,G
OOG1806P1045,GOOG1806P1047.5,GOOG1806P1050,…,
 <= 20180405 13:13:00.578 (lookup) !ENDMSG!
```

```
% Future options chain for C:
>> IQML('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,CH19
P2400,CH19P2500,CH19P2600,CH19P2700,CH19P2800,CH19P2900,CH19P3000,CH19P3100,CH19P
3200,CH19P3300,CH19P3400,CH19P3500,CH19P3600,CH19P3700,CH19P3800,CH19P3900,CH19P4
000,CH19P4100,CH19P4200,CH19P4300,CH19P4400,CH19P4500,CH19P4600,CH19P4700,CH19P48
00,CH19P4900,CH19P5000,CH19P5100,CH19P5200,CH19P5300,CH19P5400,CH19P5500,CH19P560
0,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 IQML_Chain_Callback(iqObject,eventData)
    symbols = eventData.MessageParts(2:end);  %discard the ':' message header
    % now do something useful with the reported symbols...
end
```

[146] 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);
>> IQML('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,,,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
```

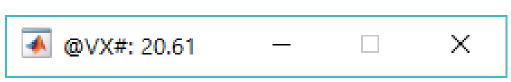
In our case, we are only interested in the 1st (Symbol) and 2nd (Most_Recent_Trade) elements of the 'Q' update messages:

```
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 above, 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 *IQML*(…,'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 *IQML* license.

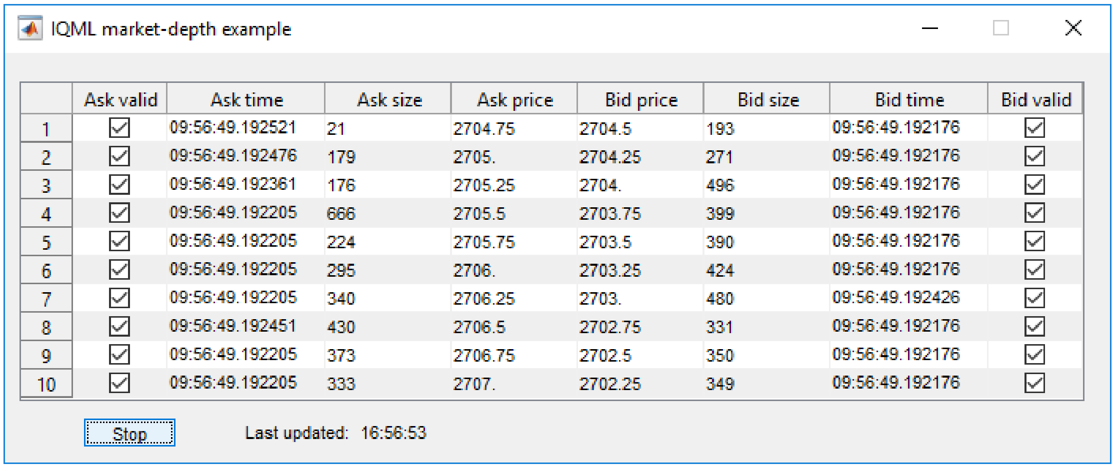
As noted in §6.4 above, market-depth events may be sent at a very high rate from the IQFeed server, and so it is not feasible or useful to update the Matlab GUI for each update. Instead, we update the GUI with the latest data at a steady rate of 2 Hz (twice a second). This can be achieved in two different ways: one alternative is to set-up a periodic timer that will run our GUI-update callback every 0.5 secs, which will call *IQML*(…,'NumOfEvents',-1) to fetch the latest data and update the GUI.

Another alternative, shown here below (also downloadable[147]), 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:

```
% IQML_MktDepth - sample Market-Depth usage function
function IQML_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','IQML 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 IQML
    contractParams = {'symbol',symbol};  % symbol='@ES#'
    callbacks = struct('Market_depth',@mktDepthCallbackFcn);
    IQML('marketdepth', contractParams{:}, 'processFunc',callbacks);
```

[147] http://IQML.net/files/IQML_MktDepth.m or https://UndocumentedMatlab.com/IQML/files/IQML_MktDepth.m

```
    % Figure close callback function - stop market-depth streaming
    function figClosedCallback(hFig, ~)
        % Delete figure window and stop any pending data streaming
        delete(hFig);
        IQML('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 % IQML_MktDepth
```



IQML market-depth example

	Ask valid	Ask time	Ask size	Ask price	Bid price	Bid size	Bid time	Bid valid
1	☑	09:56:49.192521	21	2704.75	2704.5	193	09:56:49.192176	☑
2	☑	09:56:49.192476	179	2705.	2704.25	271	09:56:49.192176	☑
3	☑	09:56:49.192361	176	2705.25	2704.	496	09:56:49.192176	☑
4	☑	09:56:49.192205	666	2705.5	2703.75	399	09:56:49.192176	☑
5	☑	09:56:49.192205	224	2705.75	2703.5	390	09:56:49.192176	☑
6	☑	09:56:49.192205	295	2706.	2703.25	424	09:56:49.192176	☑
7	☑	09:56:49.192205	340	2706.25	2703.	480	09:56:49.192426	☑
8	☑	09:56:49.192451	430	2706.5	2702.75	331	09:56:49.192176	☑
9	☑	09:56:49.192205	373	2706.75	2702.5	350	09:56:49.192176	☑
10	☑	09:56:49.192205	333	2707.	2702.25	349	09:56:49.192176	☑

Stop

Last updated: 16:56:53

11 Alerts

11.1 General Usage

In cases where certain events in steaming data are of interest to the user, *IQML* 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 *IQML* 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** [148]	colon or comma-delimited string or cell-array of strings	(none)	Limits the alert to the specified symbols and meta-tags only. Examples: • '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: • 'quote bid >= 100' • '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: [149] • 'display' • 'popup' • 'email' (requires specifying the **EmailRecipients** parameter) • @myCallbackFcn • {@myFcn, data1, data2, …}

[148] In *IQML*, the **Symbol** and **Symbols** parameters are synonymous – you can use either of them, in any capitalization

[149] Note the performance implications that are discussed in §3.6 and §10.2 above

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, *IQML* starts the data streaming automatically (see §11.2).
AlertID	integer (scalar or array)	[] (empty array)	Unique ID generated and returned by *IQML* when new alert is defined. **AlertID** is relevant (and mandatory) only for querying, editing or deletion of existing alerts. See §11.3, §11.4 below.
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: • 'john@doe.com' • 'john@doe.com, jane@doe.com' • {'john@doe.com', 'jane@doe.com'}
SmtpEmail	string	'iqml.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. **SmtpEmail** only needs to be set once, and is used by all future *IQML* alert events.
SmtpServer	string	(none)	SMTP server that will send alert emails. This parameter is relevant only for specifying a non-default email sender. **SmtpServer** only needs to be set once, and is used by all future *IQML* alert events.
SmtpPassword	string	*(confidential)*	Password of the sender's e-mail account. This parameter is relevant only for specifying a non-default email sender. **SmtpPassword** only needs to be set once, and is used by all future *IQML* alert events.

Note: Alerts are only available in the Professional *IQML* 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' or 'news')
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 ('>', '<', '=', '>=', '<=', 'contains').[150]
    - '>', '<', '=', '>=', '<=' are relevant for quote/intervalbar alerts
    - '=' and 'contains' are relevant for news alerts
4. Trigger value: either numeric (for a >,<,=,>=,<= operator) or string (for a =,contains operator)

For example:

```
alertId = IQML('alert', 'Symbol','IBM', 'Trigger','quote ask < 145', ...);
alertId = IQML('alert', 'Symbol','IBM', 'Trigger','quote Total_Volume >= 10', ...);
alertId = IQML('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 = IQML('alert', 'Symbol','IBM', ..., 'NumOfEvents',5);
```

IQML 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.

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.

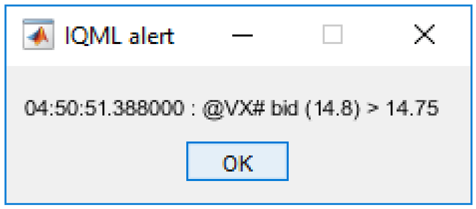
Note that with **StartStream**=`true`, the streaming is started automatically, using the default parameters. If you wish to control the streaming parameters (for example, **NumOfEvents** or **DataType**), leave **StartStream** in its default `false` value, and start the streaming in a separate *IQML* command.

[150] Additional trigger operators may be added in future *IQML* releases.

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 = IQML('alert', 'Symbol','@VX#', 'Trigger','quote bid > 14.75', ...
               'AlertAction','popup');
```



2. 'Display' announces the event in Matlab's console (Command Window):

```
alertId = IQML('alert', 'Symbol','@VX#', 'Trigger','quote bid > 14.75', ...
               'AlertAction','display');

04:50:11.099000 IQML alert: @VX# bid (14.8) > 14.75
```

   Or, as another example of regional update alert:

```
alertId = IQML('alert', 'Symbol','IBM', 'AlertAction','display', ...
               'Trigger', 'regional regionalbid > 140');

20180524 16:57:13.689 IQML 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 = IQML('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: iqml.alerts@gmail.com
Subject: IQML alert: @VX# bid (14.8) > 14.75
Body:
                          Symbol: '@VX#'
               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: iqml.alerts@gmail.com
Subject: IQML 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: iqml.alerts@gmail.com
Subject: IQML 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 *IQML* email address (iqml.alerts@gmail.com) by default. To send the alert emails from another sender (for example, a corporate email account), specify the **SmtpEmail**, **SmtpServer** and **SmtpPassword**.[151] These parameters are saved in your local machine's Matlab settings, and will be used by all future *IQML* email alerts (even after you restart the computer), so you only need to set them once. For example:

```
alertId = IQML('alert', 'Symbol','GOOG', 'Trigger','quote ask < 1090', ...
               'AlertAction','email', 'Recipients','JohnDoe@gmail.com', ...
               'SmtpServer','smtp.gmail.com', ...
               'SmtpEmail','senderEmail@gmail.com', ...
               'SmtpPassword','mypassword123');
```

On modern smartphones, text (SMS) messages have generally been replaced with push notifications of incoming emails. Still, for some users text alerts may be useful. Some mobile operators enable users to receive text messages by sending the messages to a specially-formed email address.[152] For example, to send a text message alert to T-Mobile number 123-456-7890 in the USA, simply email the alert to 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.

[151] The SMTP port is automatically assumed to be 465

[152] https://en.wikipedia.org/wiki/SMS_gateway#Email_clients. Note that carrier charges for these SMS messages might apply.

4. Callback: a personalized callback function for an event can be specified using a Matlab function handle. For example:

```
alertId = IQML('alert', 'Symbol','GOOG', 'Trigger','…', 'AlertAction',@myFunc);
```

The callback function (`myFunc` in this example) should accept two or more inputs, as customary for Matlab callbacks:[153]

```
function myFunc(alertObject, eventData)
```

- `alertObject` – a struct with the alert's configuration (see §11.3 below)
- `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: [1×1 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 = IQML('alert', 'Symbol','GOOG', 'Trigger','…', 'AlertAction',callback);

function myFunc(alertObject, eventData, data1, data2)
   ...  % data processing done here
end
```

[153] 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

IQML 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 = IQML('alert', 'Symbol','IBM', 'Trigger','quote bid > 200',…);

>> alert = IQML('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 = IQML('alert', 'NumOfEvents',-1)
allAlerts =
  3×1 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:

```
IQML('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).

```
IQML('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 IQML messages

To display detailed information on *IQML* requests and IQFeed messages, set *IQML*'s **Debug** parameter to 1 or `true` (default=0). *IQML* 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 *IQML* to IQFeed ("=>"), and incoming messages from IQFeed to *IQML* ("<="), along with the message's local timestamp and port channel:[154]

```
>> data = IQML('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 = IQML('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, *IQML* reports messages as red error messages on the Matlab console. Such messages can be handled by analyzing *IQML*'s second (optional) output argument, which is typically an empty string, except when an error is reported:

```
>> [data, errorMsg] = IQML('quotes', 'Symbol','IBM', 'Timeout',0.1)
IQML 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 =
    'IQML 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] = IQML('quotes', 'Symbol','IBM', 'RaiseErrorMsgs',false);
```

[154] Periodic IQFeed timestamp and client-stats messages (once every second) are not displayed, even **Debug** is 1 or true

In addition to IQFeed messages, your program must handle cases of *IQML* errors. In most cases, these are due to invalid *IQML* 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 *IQML* within a try-catch block, as follows:

```
try
    data = IQML('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 *IQML* 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 *IQML* are relying on default parameter values, and specifying numeric parameter values within string quotes (e.g., '1' rather than 1). Users of *IQML* should take extra precaution in their programs to ensure that these common mistakes do not occur. See discussion in §3.4 above.

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 *IQML* on a computer having more memory, or by reducing the amount of stored data.[155]

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.[156]

---

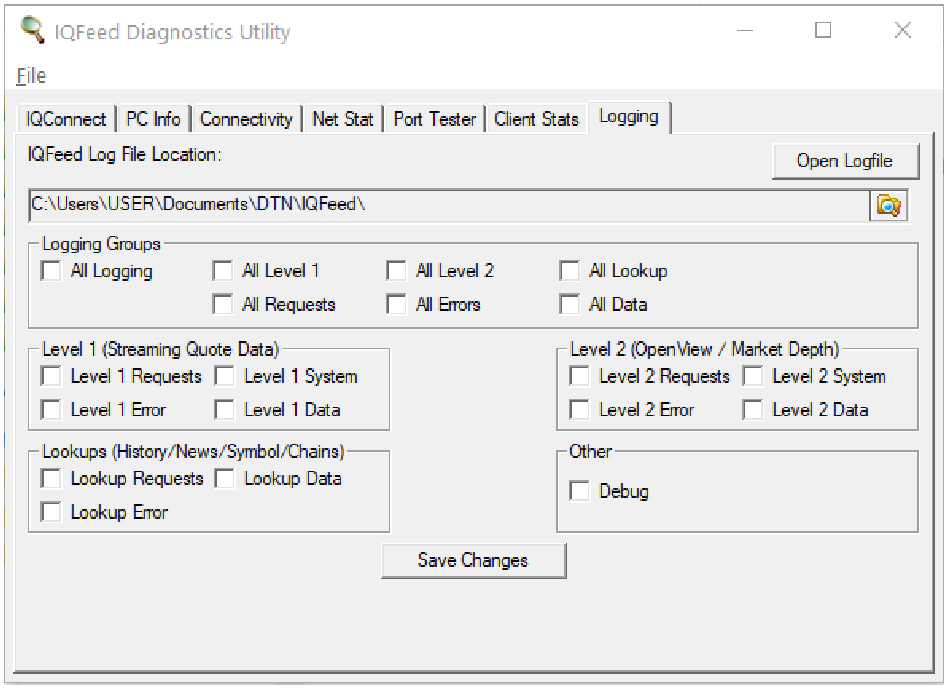
[155] Also see: http://www.mathworks.com/help/matlab/matlab_prog/resolving-out-of-memory-errors.html

[156] 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).

You can control the log file's folder path (but not its *IQConnectLog.txt* name, which is fixed), as well as the logged details, using IQFeed's Diagnostic Utility. This utility is installed as part of IQFeed's client installation:



Running `IQML('log')` without any parameters will return a Matlab struct containing the current log settings (the struct fields are explained below):[157]

```
>> data = IQML('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 *IQML*'s 'log' action:

We can modify the log file's folder path using the **Path** parameter, for example:

```
IQML('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, *IQML* will complain if the specified **Path** does not exist:

```
>> IQML('log', 'path','C:\No\Such\Folder')
Error using IQML
Bad log path specified: "C:\No\Such\Folder" is not an existing folder
```

[157] Note: the IQML('log') feature uses IQFeed's Windows registry. In Mac/Linux, IQML('log') will naturally work only when Matlab runs under Parallels/Wine, not in native mode.

When setting the log **Path**, the *IQML* 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 = IQML('log', 'path',newPath);
oldPath = fileparts(data.logFile);  % strip out the IQConnectLog.txt file name
...   % logging in this section is temporarily redirected to newPath
IQML('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 because by default, *IQConnect.exe* launches under the user's permission group and all users have full access rights to create, modify, and delete files in their user directory. Make sure your user account has appropriate permissions in the directory you choose.

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,[158] Level 1, Level 2, Lookup,[159] and Debug.[160] 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 = IQML('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 *IQML* returned the previous **Details** setting, prior to its change. A subsequent call to `IQML('log')` will verify that the **Details** change was indeed made:

```
>> data = IQML('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].

---

[158] Admin includes connection/disconnection and other non-data requests and messages

[159] Lookup includes history requests/data (§5), news (§7), and symbols/chains lookup (§8)

[160] 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 above.

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.

```
>> IQML('log', 'copyTo','C:\My Programs\Logs\log.txt');  % filename=log.txt
>> IQML('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):

```
>> IQML('log', 'doThis','reset');    % reset (empty) the live log file
>> IQML('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 *IQML* actions, we can combine different parameters in a single *IQML* command. For example:

```
IQML('log', 'details',[1 1 0 1], 'path','C:\Programs\Logs\', 'doThis','reset');
```

Here is a summary of the *IQML* parameters that affect IQFeed's internal logging:

Parameter	Data type	Default	Description
Path	string	(last-used log folder path; initially set to *My Documents\\DTN\IQFeed*)	Path of the folder in which the *IQConnect.txt* 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: • 'display' or 'show' – display the log file • 'reset' or 'clear' – empty the live log file

13 Frequently-asked questions (FAQ)

1. Can IQML be used with other data-feed providers?

IQML 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. Contact us by email to see if we can help.

2. Does IQML impose limitations on historical data or streaming quotes?

IQML does not impose any limitations, but the IQFeed server does 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 a maximum of 1000 concurrently-streaming ("watched") symbols. These limitations are imposed by the IQFeed server on your account; *IQML* supports whatever subscription your account has, it does not limit the information in any manner.

3. Can I see a demo of IQML?

You are most welcome to download a fully-functional trial version of *IQML*, to try the product at no risk.

4. How does IQML compare to alternative products?

We believe that of all the currently available alternatives for connecting Matlab to IQFeed, *IQML* provides by far the best functionality, value and cost-effectiveness. You are most welcome to test this yourself, using *IQML*'s free trial.

5. Does IQML come with an IQFeed or market subscription?

No – *IQML* connects to an existing IQFeed account. You will need to purchase the IQFeed and market subscriptions separately from DTN.

6. Does IQML send you any information?

No – *IQML* only communicates with IQFeed. The only communication that is done with *IQML*'s server is a verification of the license activation (a single hash-code).

7. How can I be sure IQML does not contain bugs that will affect my trades?

Well, there is never a 100% guarantee. The product is rigorously tested. To date, nothing major has been reported by users. *IQML* is rock solid - a very stable and robust product.

8. Is IQML 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 *IQML*'s change-log, listed in Appendix B of the *IQML* User Guide (this document). You can see the very latest updates in the online version of this guide.[161]

[161] http://IQML.net/files/IQML_User_Guide.pdf or https://UndocumentedMatlab.com/IQML/files/IQML_User_Guide.pdf

9. I saw a nice new feature in the online User Guide – can I get it?

Once you install *IQML*, 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, directly from the Matlab console, as follows:

```
>> IQML('update')
Downloading the latest IQML version from http://IQML.net/files/IQML.zip
 into C:\IQML\...
Download complete - installing...
Installation of the latest IQML version is now complete.
Please restart Matlab for the new version to take effect.
```

In addition, you can always download the latest version of *IQML* at any time from http://IQML.net/files/IQML.zip or https://UndocumentedMatlab.com/IQML/files/IQML.zip.

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 IQML license will expire in 3 days (10-Mar-2018).
*** To extend your license please email info@IQML.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, *IQML* will stop working. You can always renew or extend your license using the payment links on http://IQML.net or https://UndocumentedMatlab.com/IQML. If you wish to be independent of such annual renewals, you can select a discounted multi-year license.

11. Can I transfer my IQML license to another computer?

Yes, simply email us and we will make the activation switch for you. At any one time, each *IQML* 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 IQML on both?

Yes, but you will need to purchase two separate *IQML* licenses. *IQML*'s license is tied to a specific computer, unless you purchase a site license.

13. Can IQML be compiled and deployed?

Yes, *IQML* can indeed be compiled. You do not need a separate license for the compiled application on your development computer, since this computer is already licensed. However, any other deployed computer will require a separate *IQML* license, otherwise *IQML* will not run. If you wish to deploy *IQML* on a large scale, to multiple end-user computers, contact us to discuss alternatives.

14. Is IQML provided in source-code format?

IQML 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 IQML'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's[162] 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 IQML?

IQML supports the entire IQFeed API. This means that all the functionality that IQFeed exposes in its API, is available in *IQML* 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 *IQML*), please refer to *IQML*'s User Guide (this document), IQFeed's online reference, or contact IQFeed customer service.

17. Can you add feature ABC in IQML 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 *IQML*, 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 *IQML* features, strategy development will never be disclosed to others, and will not be integrated in *IQML*. It will be developed privately for you, and will be kept secret. See §15 below for additional details. If you have such a request, contact us by email to get a proposed quote.

19. Does IQML include back-testing/charting/data analysis/algo-trading?

No. *IQML* 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 *IQML*. We have extensive experience in developing complete backtesting and real-time trading applications - see §15 below for additional details. We will be happy to either develop a new application based on your specifications, or to integrate *IQML* into your existing application, under a separate consulting contract.

[162] http://nccgroup.com/en/our-services/software-escrow-and-verification/software-escrow

14 Troubleshooting

Error	Description / solution	Section
NullPointerException com.mathworks.jmi.bean.MatlabBeanInterface.-addCallback	*IQML* cannot work properly unless its Java file (*IQML.jar*) is added to Matlab's static Java classpath. Contact us to solve the problem.	§2.1
IQFeed is not properly installed	IQFeed is not installed properly on the local computer so *IQML* cannot connect to it.	§2.1
IQFeed cannot be connected or started or: Cannot connect to IQFeed	*IQML* cannot connect to an active (running) IQFeed client process, nor start one. Try to start IQFeed's client manually and then retry.	§2.1
IQML 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 IQML license will expire in 4 days (1-Mar-2018)	This is an alert on upcoming license expiration. It is not an error, and does not affect *IQML*'s operation. Contact us to extend your license.	§2.2
Your IQML license has expired on 1-Jun-2018	*IQML*'s license is limited in duration. When the license term expires, contact us to renew it.	§2.2
Cannot connect to IQML.net to validate your IQML license	*IQML* validates its license on the *IQML* server. Your internet connection may be down, or the domain (iqml.net, undocumentedmatlab.com) 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 *IQML* action, although it is planned for a future version.	§2.4
Unrecognized IQML action 'xyz'	The specified action is invalid in *IQML*. 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. *IQML* 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 *IQML* 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 IQML	The specified parameter name is not valid for the specified *IQML* 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 IQML	The specified action is only available in the *IQML* 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 IQFeed.	§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: IQML 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 *IQML* 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
`IQML 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 *IQML* 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
`IQML historic data query error: !NO_DATA!`	No data is available for the specified query. Try to modify the query parameters.	§5
`Symbol "XYZ" is not currently streaming`	Start data streaming (by sending a query with **NumOfEvents**>0) before querying streamed data	§6
(*IQML* stops receiving IQFeed streaming data)	Try to actively disconnect and reconnect to IQFeed, or to restart the *IQConnect* 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)	*IQConnect* lost the connection to IQFeed's servers. *IQConnect* 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 *IQML* 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 *java.opts* file.	§12.1

15 Professional services

In addition to *IQML* being offered as an off-the-shelf software program, advanced Matlab consulting, training, and program development are being offered. With close to 30 years of professional Matlab programming experience, including extensive finance/trading-related development in the past decade, we offer top-of-class Matlab consulting, with a particular emphasis on the financial sector.

In particular, we have a lot of experience integrating quality production-grade Matlab programs with online brokers (e.g. Interactive Brokers (IB), CQG, CFH FIX), data-feed providers (e.g. DTN IQFeed, Bloomberg, Reuters, Trading Physics), websites (e.g. Finviz, Nasdaq), databases (e.g. SQL Server, Oracle, MySQL, SQLite), as well as Excel and raw-format data files. Programs were developed on multiple Matlab releases and all the Matlab-supported platforms: Windows, Mac and Linux.

We have completed countless life-cycles of software requirements definition, design, development, documentation, integration, testing, deployment, handover, maintenance and support.

Much of our work derives from the financial sector: We developed custom software for a commodities fund in a Geneva bank; a backtesting and analysis program for a large bank in Chicago; a currencies trading program for a hedge-fund in Malta; data-analysis products for financial services firms in New-York; a portfolio risk/exposure analysis program for an Israeli investment advisor; a charting GUI for a San-Francisco hedge fund; and semi- and fully-automated algo-trading programs for multiple clients around the globe.

Development is typically done remotely; onsite consulting/development is also possible upon request, or a combination of remote work and onsite visits.

You can see a small sample of programs that we have developed below. Additional samples can be seen on our consulting webpage.[163]

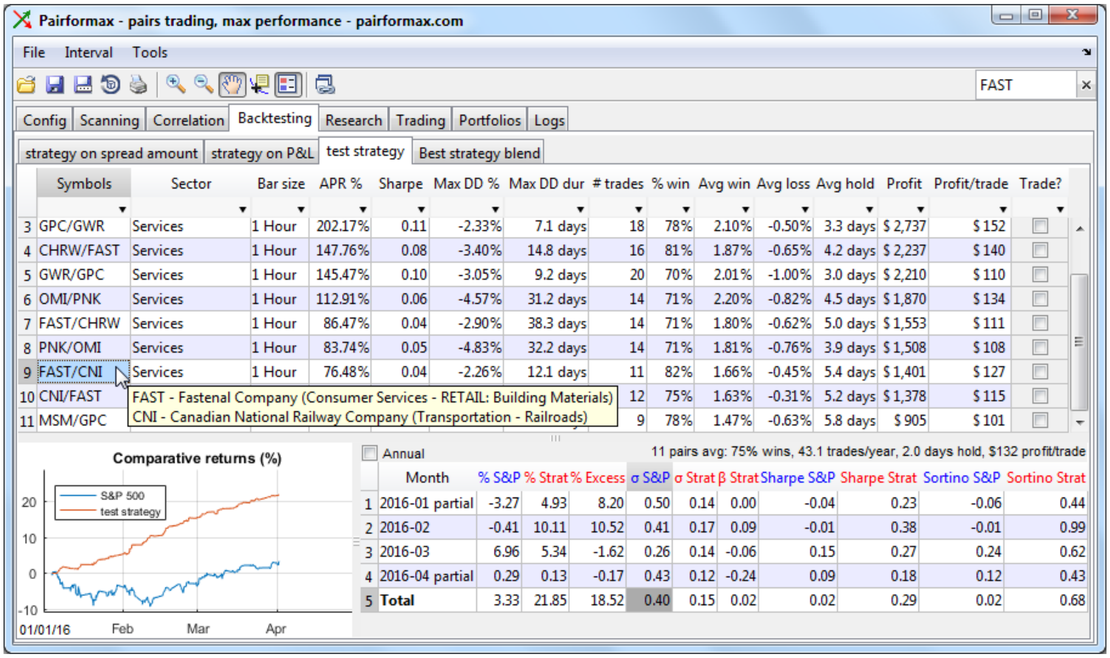
Anything developed under private consulting will be kept confidential and will not be disclosed to others. You will retain full IP ownership of anything developed for you.

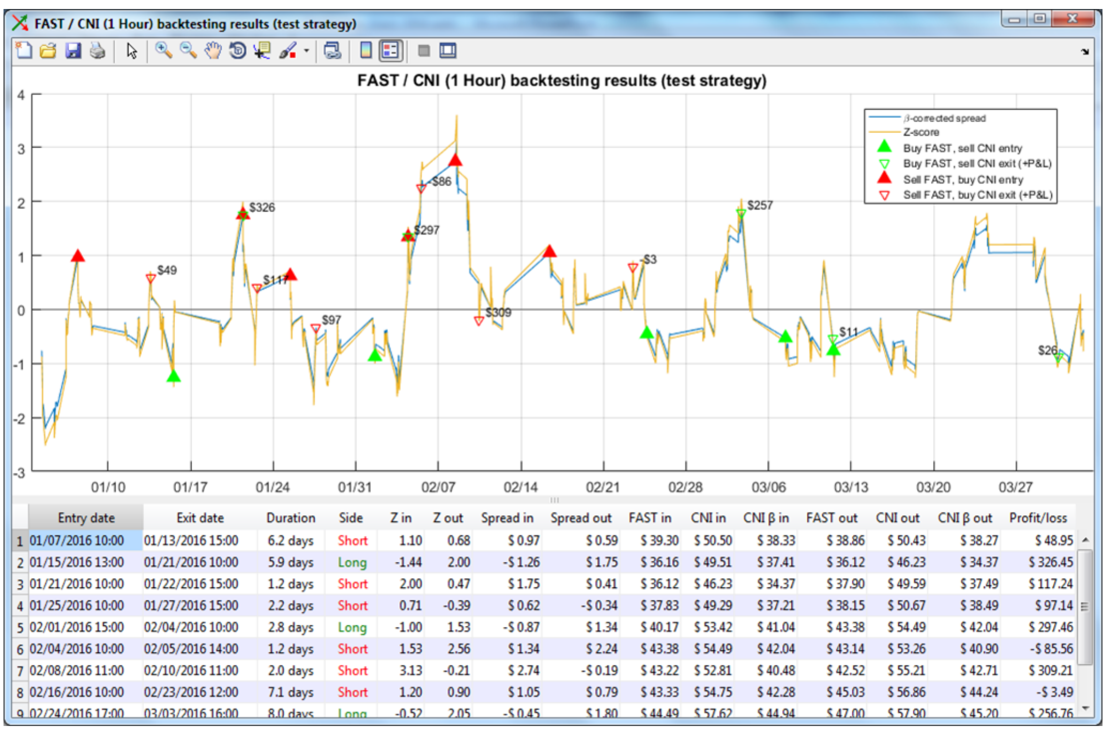
Most of our revenue comes from returning or long-time clients. We will be happy to provide references of satisfied clients in US or Europe. With such an impressive track record, you probably already know some of them.

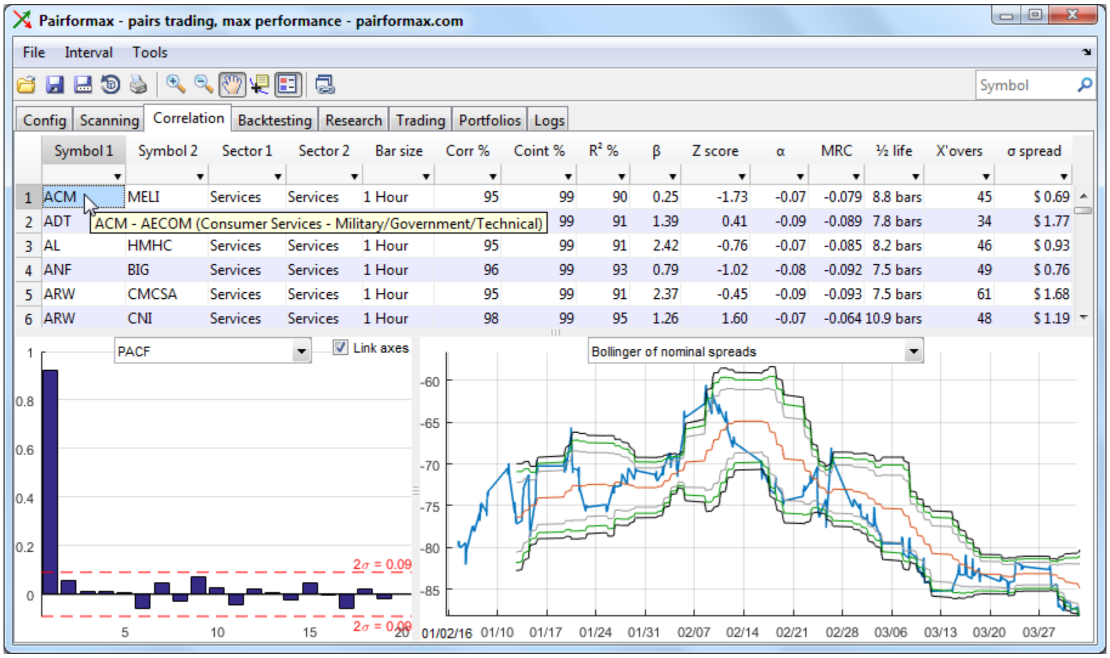
Contact us by email (info@IQML.net or info@UndocumentedMatlab.com) to discuss your needs or to receive a proposal.

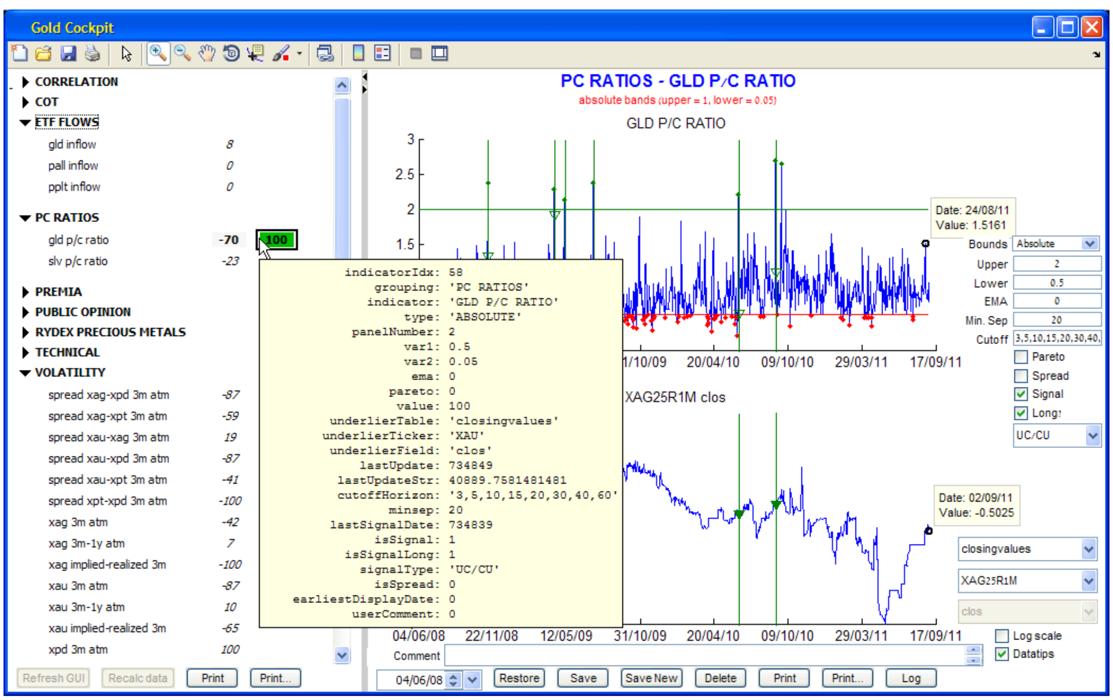
[163] http://undocumentedmatlab.com/consulting

15.1 Sample program screenshots

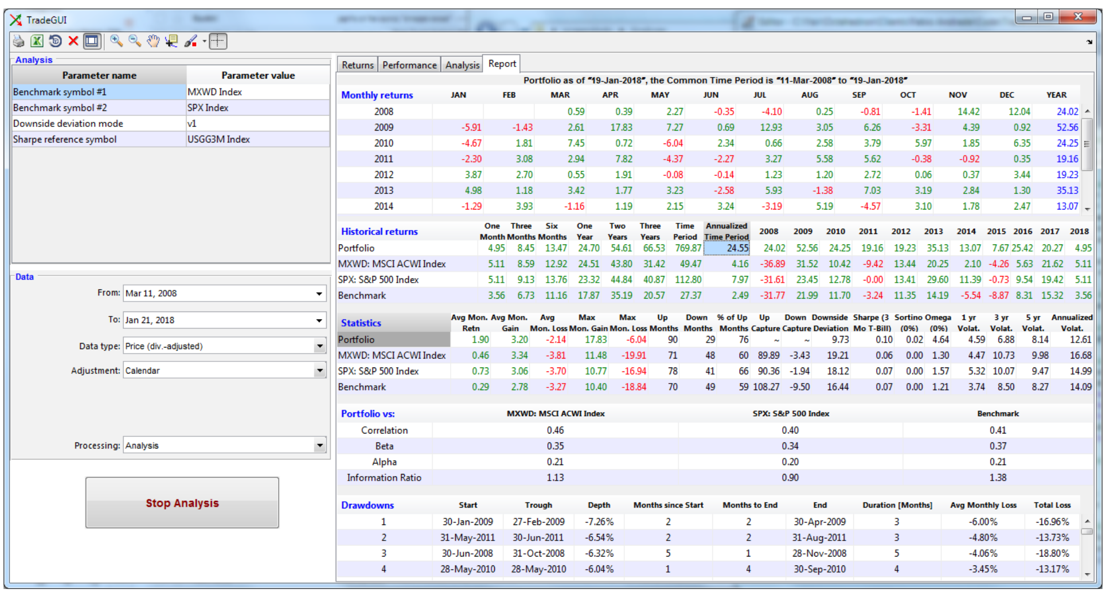





Pairformax - pairs trading, max performance - pairformax.com
File Interval Tools
Config Scanning Correlation Backtesting Research Trading Portfolios Logs
Symbol 1 Symbol 2 Sector 1 Sector 2 Bar size Corr % Coint % R² % β Z score α MRC ½ life X'overs σ spread
1 ACM MELI Services Services 1 Hour 95 99 90 0.25 -1.73 -0.07 -0.079 8.8 bars 45 $ 0.69
2 ADT ACM - AECOM (Consumer Services - Military/Government/Technical) 99 91 1.39 0.41 -0.09 -0.089 7.8 bars 34 $ 1.77
3 AL HMHC Services Services 1 Hour 95 99 91 2.42 -0.76 -0.07 -0.085 8.2 bars 46 $ 0.93
4 ANF BIG Services Services 1 Hour 96 99 93 0.79 -1.02 -0.08 -0.092 7.5 bars 49 $ 0.76
5 ARW CMCSA Services Services 1 Hour 95 99 91 2.37 -0.45 -0.09 -0.093 7.5 bars 61 $ 1.68
6 ARW CNI Services Services 1 Hour 98 99 95 1.26 1.60 -0.07 -0.064 10.9 bars 48 $ 1.19
PACF
Link axes
Bollinger of nominal spreads


Gold Cockpit
CORRELATION
COT
ETF FLOWS
gld inflow 8
pall inflow 0
pplt inflow 0
PC RATIOS
gld p/c ratio -70 100
slv p/c ratio -23
PREMIA
PUBLIC OPINION
RYDEX PRECIOUS METALS
TECHNICAL
VOLATILITY
PC RATIOS - GLD P/C RATIO
GLD P/C RATIO
XAG25R1M clos
Date: 24/08/11
Value: 1.5161
Date: 02/09/11
Value: -0.5025
Bounds Absolute
Upper 2
Lower 0.5
EMA 0
Min. Sep 20
Pareto
Spread
Signal
Log scale
Datatips
Comment
Refresh GUI Recalc data Print Print...
Restore Save Save New Delete Print Print... Log


TradeGUI
Analysis
Parameter name
Parameter value
Benchmark symbol #1
MXWD Index
Benchmark symbol #2
SPX Index
Downside deviation mode
v1
Sharpe reference symbol
USGG3M Index
Data
From:
Mar 11, 2008
To:
Jan 21, 2018
Data type:
Price (div.-adjusted)
Adjustment:
Calendar
Processing:
Analysis
Stop Analysis
Returns
Performance
Analysis
Report
Distribution of total returns (monthly %)
Number of months
Risk vs. return
Annualized return
Annualized volatility
Rolling returns
Total return
Worst
Best
1 month
3 months
6 months
12 months
24 months
36 months
60 months


TradeGUI
Analysis
Parameter name
Parameter value
Benchmark symbol #1
MXWD Index
Benchmark symbol #2
SPX Index
Downside deviation mode
v1
Sharpe reference symbol
USGG3M Index
Data
From:
Mar 11, 2008
To:
Jan 21, 2018
Data type:
Price (div.-adjusted)
Adjustment:
Calendar
Processing:
Analysis
Stop Analysis
Returns
Performance
Analysis
Report
Monthly returns
Historical returns
Statistics
Portfolio vs:
Correlation
Beta
Alpha
Information Ratio
Drawdowns

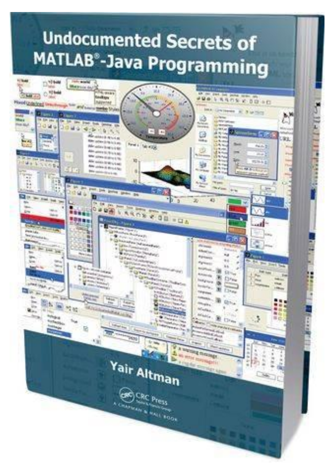
15.2 About the author

With close to 30 years of professional software programming experience, Yair Altman offers top-notch Matlab consulting and training services.

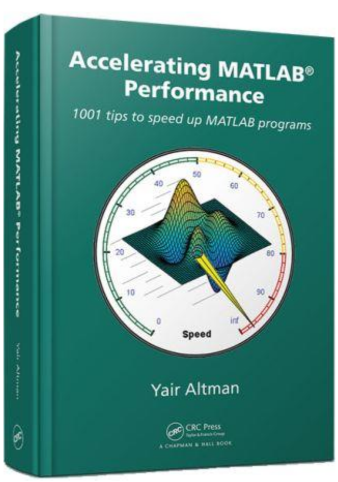
Yair has worked extensively with Matlab and many other programming languages (Java, C#, C, C++ and others). He has developed many programs with SQL and a variety of databases, operating systems and hardware platforms.

Matlab community developers, and even MathWorks themselves, consider Yair a top Matlab expert, as any simple online search will show. His UndocumentedMatlab.com website is by far the largest and most active independent Matlab site. Yair is also well-known from numerous submissions on the Matlab forums and File Exchange; a MathWorks study determined[164] that Yair is the third most influential submitter in the entire Matlab community. He regularly advises MathWorks, and is a member of both its *Community Advisory Board* (CAB) and *MATLAB Advisory Board* (MAB).

Yair has a specific experience in the finance sector, developing quality professional Matlab programs that integrate with trading platforms (IB, CQG, CFH FIX); data-feed providers (DTN IQFeed, Bloomberg, Reuters, Trading Physics); websites (Finviz, Nasdaq); databases (SQL Server, Oracle, MySQL, SQLite), as well as Excel and raw-format data files. These programs were developed on multiple Matlab releases and all Matlab-supported platforms: Windows, MacOS and Linux.



Yair published two extensive and highly-acclaimed Matlab textbooks: *MATLAB-Java programming*[165] (2011) and *Accelerating MATLAB Performance*[166] (2014). Both books are considered the top references in their field.



Yair provides professional, cost effective consulting and contract development.[167] He can do stuff that few other Matlab programmers know is even possible, delivering great value: top quality code at reasonable cost.

Yair also offers custom Matlab training courses,[168] in a variety of topics and levels.

By combining a proven track-record of quality software programming, decades of professional experience, and Matlab knowledge that few others possess, Yair provides clients with superior value and cost-effectiveness.

[164] http://blogs.mathworks.com/community/2013/01/15/giving-by-taking-file-exchange-acknowledgment-trees

[165] http://undocumentedmatlab.com/books/matlab-java

[166] http://undocumentedmatlab.com/books/matlab-performance, now in 2nd edition

[167] http://undocumentedmatlab.com/consulting

[168] http://undocumentedmatlab.com/training

Appendix A – online resources

A.1 Official DTN IQFeed resources

- IQFeed homepage – http://iqfeed.net
- IQFeed API homepage – http://www.iqfeed.net/dev/api/docs
- IQFeed symbol guide – http://iqfeed.net/symbolguide/index.cfm?symbolguide=guide&displayaction=support&section=guide&web=iqfeed
- IQFeed symbol lookup – http://iqfeed.net/symbolguide/index.cfm?symbolguide=lookup&displayaction=support&section=guide&web=iqfeed
- IQFeed users forum – http://forums.iqfeed.net
- IQFeed live chat – http://iqfeed.net/Fibonacci/index.cfm?displayaction=support&section=chat
- API customer service and technical support – support@iqfeed.net or http://iqfeed.net/Fibonacci/index.cfm?displayaction=support&section=contact (please let them know that you are using *IQML*)

A.2 MathWorks webinars/presentation

- MathWorks algorithmic-trading portal – http://mathworks.com/discovery/algorithmic-trading.html, http://mathworks.com/financial-services/algorithmic-trading.html (includes Yair's webinar "*Real-Time Trading System in MATLAB*")
- Algorithmic Trading Strategies with MATLAB Examples – https://mathworks.com/videos/algorithmic-trading-strategies-with-matlab-examples-92899.html
- Energy Trading & Risk Management with MATLAB – https://mathworks.com/videos/energy-trading-risk-management-with-matlab-81745.html
- Cointegration and Pairs Trading with the Econometrics Toolbox – https://mathworks.com/videos/cointegration-and-pairs-trading-with-econometrics-toolbox-81799.html
- Commodities Trading with MATLAB – https://mathworks.com/videos/commodities-trading-with-matlab-81986.html
- Creating professional-quality applications with MATLAB – (Yair's keynote presentation in the 2016 Munich MATLAB Expo using IQFeed) https://undocumentedmatlab.com/blog/upcoming-public-matlab-presentations

A.3 Additional open-source Matlab resources

- Spatial Econometrics Toolbox for Matlab – http://spatial-econometrics.com
- Algorithmic trading code in the Matlab File Exchange – http://www.mathworks.com/matlabcentral/fileexchange/?term=trading

Appendix B – change log

The following table lists changes done to this document and *IQML*. Depending on the date that you have installed *IQML*, your version may be missing some features discussed in this document. You can always update to the latest version – see §2.4.

B.1 Complete change log (functional + documentation)

The table below lists both functional changes (in the *IQML* program) and also documentation changes (in this User Guide). See §B.2 below for a table of only the functional change.

* In this table, the last column indicates the change type: **F**=functional; D=documentation:

Version	Date	Section	Description	*
0.80	2017-10-17	-	Beta integration of *IQML* in a user trading program	**F**
1.00	2018-02-26	-	First commercial release of *IQML*	**F**
1.01	2018-03-11	8.1	Enabled message-specific user callbacks; Added additional information to callback eventData	**F**
1.02	2018-03-12	4.3.2	Clarified filtering meta-symbols such as 'BZRatings'	D
		4.3.3	Added relevant symbols list in returned news story data	**F**
		7.1, 11	Clarified automatic connection re-establishment	D
1.03	2018-03-19	3.2	Enabled **Symbol** and **Symbols** as synonymous params	**F**
		4.1, 6.1	Improved ticks request logic & the returned data fields	**F**
		6.1	Enabled requesting streaming ticks/quotes for multiple symbols at once, in a single *IQML* command	**F**
1.04	2018-04-01	1, 2.1	Clarified that IQFeed client can run on Linux/Mac via Parallels/Wine, as well as natively on Windows/Mac	D
		2.1	Added support for native Mac IQFeed client (untested)	**F**
		3.2	Added new **MsgParsingLevel** general parameter, for improved callback run-time performance	**F**
		5	**MaxDataItems** input parameter is renamed **MaxItems**	**F**
		6.1	Some result output fields renamed for consistency; **BufferSize** input parameter is renamed **MaxItems** for consistency; clarified the documentation text	**F**
		6.2	Added new streaming regional updates functionality	**F**
		4.3→7	Moved the news functionality into a new chapter (§7)	D
		7.3	Added newline characters between separate paragraphs in the reported news-story text, for better readability	**F**
		7.4	Clarified that default **Date** is today; clarified that story count also includes non-subscribed news sources	D
		7.5	Added new streaming news functionality	**F**
		8-13	Renumbered chapters 7-12 as 8-13, to make room for the new chapter (§7) on the news functionality	D
		8.2	Added new section on callback run-time performance	D
		10	Added timestamp and channel info to debug printouts	**F**
		A.2	Added an online MathWorks resource	D

Version	Date	Section	Description	*
1.05	2018-04-05	2.1	Added note that in some cases users may need/want to specify the IQFeed connection **Username**, **Password**	D
		3.1, 8	Added new symbols and numeric market codes lookup functionality	F
		3.2, 4.1 5, 7.2	Modified the default Timeout value from 3 to 5 [secs]	F
		3.5	Added new section on handling returned data format	D
		5.5	Clarified that micro-sec time resolution depends on the IQFeed client version, the market, and the security type	D
		8.2, 10.3	Added basic support for options-chain and futures-chain symbol lookup (better support is planned for next version)	F
		9-14	Renumbered chapters 8-13 as 9-14, to make room for the new chapter (§8) on the lookup functionality	D
		9.1	Enabled specifying IQFeed **Username** and **Password**; Added a 10-sec timeout on IQFeed connection attempts	F
		9.3	Added extra port-specific stats when **AddPortStats**=1	F
1.06	2018-04-08	8.2	Added options/futures chain lookup functionality	F
1.07	2018-04-10	9.1	Added info msgs on server connections/disconnections	F
1.08	2018-04-11	4.1, 6.1	Added the Symbol field to returned quotes data struct	F
		10.4	Added usage example of realtime quotes user callback	D
1.09	2018-04-16	6.3	Added Interval Bars functionality	F
1.10	2018-05-04	2.4	Added example of update notification on a new version	D
1.11	2018-05-16	4.3, 6.4, 10.5	Added Market Depth (Level 2) functionality	F
		6.3	Indicated that IQFeed server may possibly limit reported interval bars depending on exchange, data subscriptions; Clarified that **IntervalSize** must be >1 for volume/ticks	D
1.12	2018-05-23	3.4, 4.3, 6.2, 6.4, 7, 8.2, 10.5, 12	Clarified that news, level 2 (market depth), alerts, options/futures chain lookup, and regional updates are only available in the Professional license and free trial	D
		12	Added alerts functionality	F
		13-15	Renumbered chapters 12-14 as 13-15, to make room for a new chapter on the alerts functionality	D
1.13	2018-05-25	7.2	Enabled auto-fetch of full news story in news headlines query (streaming/blocking) using **GetStory** parameter	F
		11-12	Switched between sections 11,12 in the User Guide	D
		11.2	Enabled reporting the full news story (in addition to headline) in news alerts using **GetStory** parameter	F
		11.1, 11.2	Added regional updates alert functionality (in addition to news/quote/intervalbar alerts)	F
1.131	2018-05-28	3.1	Fixed bug in accepting struct-based input parameters	F
		5.4	Clarified that **IntervalSize** must be >1 for vol/tick bars	D
		6.2	Fixed typo in regional update action (should be 'regional')	D
		7.2	Fixed bug in the news headlines functionality	F

Version	Date	Section	Description	*
1.14	2018-05-30	4.2	Enabled specifying multiple **Symbols** in a single Fundamental-data query	F
		6.2	Enabled specifying multiple **Symbols** in a single streaming Regional updates query	F
		7.3	Enabled specifying multiple news headline **ID** values in a single news story query	F
1.15	2018-07-08	Cover	Updated compatibility notice for Matlab release R2018b	D
		4.1, 6.1, 14	Enabled querying snapshot (top of market) & streaming data of multiple symbols at once, in a single *IQML* query	F
		4.2	Fixed: querying multi-symbol fundamental data sometimes returned empty results	F
		6.1-6.3	Fixed: debug data was displayed when streaming queries were requested (now only displayed if **Debug**=1)	F
		8.2	Enabled querying fundamental data of all symbols in an options/futures chain at once, in a single *IQML* query	F
		8.2	Enabled querying snapshot (top of market) data of entire options/futures chain at once, in a single query	F
		9.1	Fixed: *IQML* query during IQFeed connection sometimes returned empty/error results	F
1.16	2018-07-09	3.6	Added new section on general run-time performance	D
		5	Improved performance (speed) of historical data queries	F
		10.2	Updated the section on callback-related performance	D
1.17	2018-07-30	5.4, 5.5	Clarified that IQFeed limits ticks/interval data to 8 days during US trading hours, 180 calendar days outside them	D
		6.1	Clarified that IQFeed allows up to 500 concurrently-streaming symbols, unless you pay DTN for more symbols	D
		6.3	Clarified that **IntervalSize** must be >1 for interval bars that use **IntervalType** = 'ticks' or 'volume'	D
		6	Enabled retrieval and cancellation of streaming data for multiple/all streamed symbols in a single *IQML* command	F
		8.2	Clarified that option/future chain name might change when corporate actions (such as splits) occur	D
1.18	2018-08-03	3.1, 3.5	Added optional errorMsg output for *IQML* commands	F
		9.1	Fixed problem of duplicate fields during initial connection	F
		9.1	Improved the reliability of a programmatic IQFeed disconnect/reconnect	F
1.19	2018-08-06	3.2, 12	Added the **RaiseErrorMsgs** parameter to control whether IQFeed errors should raise a Matlab error	F
		4.1, 5.1, 7.2, 8.1	Message about partial data received due to **Timeout** is now a Matlab warning message, not an error message	F
1.20	2018-08-07	5, 14	Enabled requesting history data for multiple symbols in a single *IQML* command	F
		5.1, 5.4, 5.5	Automatically convert **BeginDate**⇔**BeginDateTime**, **EndDate**⇔**EndDateTime** (i.e. try to fix usage error)	F

Version	Date	Section	Description	*
1.21	2018-08-10	8.1	Enabled looking up symbols by market(s), sec-type(s)	F
		8.2	Clarified that IQFeed only enables lookup of active (non-expired) options; a list of expired options is available separately as a downloadable text file.	D
1.22	2018-08-13	8.2	Enabled **NearMonths** values of 0-12, not just 0-4, for options/futures chain. Note: this is based on undocumented IQFeed functionality, so might not work in some cases.	F
		3.2 etc.	Limited the **Timeout** parameter values to 0-3000 [secs]	F
1.23	2018-08-14	5.4	Clarified regarding historical intervals data limitations; Clarified that IQFeed's interval data typically exclude irregular "O" trades (see §5.5).	D
		9.1	Fixed a problem of possible bad connection to IQFeed during the initial connection by *IQML*	F
1.24	2018-08-31	3.2 etc.	Limited the **Timeout** parameter values to 0-9000 [secs], with 0 indicating infinite (i.e. no-limit) timeout	F
		5.5	Clarified that while IQFeed typically limits historic tick data to 180 days (outside trading hours), extended (older) tick data can possibly be purchased from DTN	D
		8.2	Enabled **NearMonths** values of 0-99, not just 0-12, for options/futures chain. Note: this is based on undocumented IQFeed functionality, so might not work in some cases.	F
		9.1	Enabled multiple Matlab processes on the same computer to run *IQML* concurrently (Beta)	F

Version	Date	Section	Description	*
2.00 (major update)	2018-09-05		*This is a major update. Highlights: query parallelization and multiple usability/functionality fixes/improvements*	
		(all)	Enabled parallel processing of *IQML* commands within `parfor`/`spmd` blocks, and parallel internal processing via the **UseParallel** parameter (Professional license only)	F
		2	Added the license type to the output of *IQML('version')*	F
		3.1	Clarified the actions available in Pro vs. non-Pro license	D
		3.5	All returned data arrays are now column vectors	F
		3.5	Using the 2nd (optional) output of *IQML* (`errorMsgs`) now implies a default value of `false` for **RaiseErrorMsgs**	F
		3.5, 8.2-8.7	Fixed various typos in code snippets, that would have resulted in errors or bad data if used as-is	D
		4	Modified reported data format when **NumOfEvents** >1	F
		4.1, 8.2	Issued a warning when requesting more symbol quotes than your IQFeed account limit	F
		4.3	Added new section on blocking interval bars functionality	F
		4.3, 5.4, 6.3	Clarified that **IntervalSize** must be ≥100 for volume bars (a new limitation of IQFeed)	F
		4.3, 5, 6.3, 7.4	Enabled specifying dates and date-times using Matlab `datetime` objects (in addition to datenums and strings)	F
		4.3→4.4	Moved the blocking market-depth section to §4.4	D
		5.1, 5.4, 5.5	Clarified that **MaxItems** has precedence over **BeginDate /Time** when more data items are available than **MaxItems**	D
		5.4, 5.5	Clarified that in IQFeed and *IQML*, 'ticks' = 'trades'	D
		6	Added `Symbol` field to returned streaming data struct	F
		7.4	Story count for symbols that have no related news story is reported as 0 (such symbols were previously skipped)	F
		9.3	Added `Exchanges`, `ServerVersion`, `ServiceType` fields to the returned client stats data	F
		11	Fixed various things with the Alerts functionality	F
		11.2	Reorganized & clarified the Alerts Configuration section	D
2.02	2018-09-13	4.3, 5.4, 6.3	Clarified that **IntervalSize** must be < 86400 for secs bars (a new limitation of IQFeed); added warning when user attempts to use an invalid **IntervalSize** value.	F
		4.3, 6.3	Clarified that streaming/latest interval bars are subject to the same limitations as those imposed on historical bars	D
		5.4	Clarified that full-minute interval bars are exempt from the 8/180-day limitation imposed by IQFeed's servers	D
		9.1	Added detection & report for a case of a non-communicative background IQConnect process	F
2.03	2018-09-30	9.1	Fixed a problem with the license check that caused IQFeed disconnections	F
2.04	2018-10-02	5	Improved download speed of historical data queries	F
		6, 7.5, 9.2	`LatestEventTimestamp` is now reported in seconds (not msecs) resolution by default, unless **Debug** is 1 or `true`	F

Version	Date	Section	Description	*
2.05	2018-10-13	4.1, 6.1	Added **Fields** parameter to enable dynamic fields-set	F
2.06	2018-10-15	4.1	Added some clarifications on the new **Fields** parameter	D
		6.1	Minor fixes, performance speedup of streaming quotes	F
2.07	2018-10-21	3.4	Minor text clarifications; added timestamp examples	D
		4.3, 5.4, 6.3	Clarified that IQFeed's limitations on live 'secs' interval bars are stricter than limitations on historical intervals	D
		5.4	Enabled using **MaxDays** as synonym for the **Days** parameter in historic interval queries	F
		9.1	Fixed a problem with the license validation that prevented connection in certain cases	F
		9.4	Fixed a few small edge-cases with sending custom commands to IQFeed	F
2.08	2018-10-28	3.6	Added clarifications on the use of query parallelization	D
		5.1, 5.4, 5.5	Enabled parallelized historic data queries (daily/interval/ticks) that have date/time range (Professional license only)	F
		A.1	Added IQFeed's users forum to list of online resources	D
2.09	2018-11-07	3.1	Fixed a bug in parsing input parameters in struct format	F
		3.6	Added explanation on how to use a customized **Fields** parameter to improve the query speed of market quotes	D
2.10	2018-11-14	2.4	Added ability to revert back to the previous *IQML* version at any time.	F
		4.1	Added a table listing all the available quote data fields (customizable via the **Fields** parameter)	D
		4.1	Added description fields for the `Bid_Market_Center`, `Ask_Market_Center` and `Last_Market_Center` fields, when reported in a quotes message from IQFeed.	F
		4.5	Added a new `'Greeks'` action, to calculate Greeks, fair value price and implied volatility for options (Professional license only)	F
		5.1	Clarified that DTN limits historical data retrieval in IQFeed's trial account. Historical data queries in such accounts may yield fewer data points than requested.	D
		6.1	Clarified that tick (update/quote) messages are streamed whenever any of the requested **Fields** gets updated.	D
2.11	2018-11-22	4.5	**DaysPerYear** parameter was renamed **AnnualFactor**; **Duration** parameter was renamed **DaysToExpiration**; `Vega`, `Rho`, `Veta`, `Ultima` are no longer divided by 100 (compatibility with Matlab Financial Toolbox, Maple & NAG); minor fix for `Veta` (negative value); Added new fields in the reported data struct: `Omega` + `Lambda` (synonyms), `CRho`, `Color`, `Annual_Factor_Used`. Clarified differences of *IQML*'s Greek values vs. Matlab's Trading Toolbox, NAG, and Maple. Added a table explaining all the reported Greek values.	F

Version	Date	Section	Description	*
2.12	2019-01-16	2.2	Added cross-check for *IB-Matlab* connector	F
		2.2	Clarified some license variants; mentioned *IB-Matlab* bundle	D
		2.4	Added a new `'revert'` action, to revert back to a previous *IQML* version	F
		3.1	Fixed: display this User Guide using *IQML*('doc') even when the document is not on the Matlab path	F
		4.5	Clarified that Greek Vomma is sometimes called Volga; minor clarifications regarding the foreign (carry) rate.	D
		6.2	Clarified the functionality of streaming regional updates	D
2.13	2019-02-28	2, etc.	Clarified that https://UndocumentedMatlab.com/IQML can be used interchangably with http://iqml.net for any *IQML* document or file resource	D
		2.1	Updated licensing alternatives (short-term, bundle); mentioned option of using the Matlab `pathtool` command	D
		2.2	Updated and clarified the license reactivation process	D
		3.4	Clarified that *IQML*'s timestamp data fields use either local or New York time, not the exchange time	D
		3.6, 4.5	Clarified documentation, improved readability	D
		6.1-6.3	Added the **ClearBuffer** parameter for streaming data; fixed bug with streaming data when **NumOfEvents**=inf; fixed the documentation of **NumOfEvents** default value	F
		6.4	Clarified that the streaming market depth mechanism does not store an internal buffer of quote updates	D
		13.15	Clarified FAQ #15 on business continuity alternatives	D
2.14	2019-03-14	3.1	Fixed a problem specifying parameters via Matlab table	F
		3.1	Added ability to specify parameters using Matlab class object properties, similar to struct fields	F
		3.1	Removed misleading reference to struct array (only a scalar struct is currently supported, not a struct array)	D
		3.1, 14	Invalid parameter names are now ignored (a warning is displayed), rather than raising a Matlab error/exception	F
		3.4	Clarified IQFeed's sensitivity to valid symbol names	D
		4.1 etc.	Improved handling of multi-symbol queries	F
		4-7, 11.1	**Symbols**, **Fields**, and news **Sources** can now be separated by ':' or ',' (for example, 'IBM:HP' or 'IBM,HP')	F
		6.1	Clarified that the TST$Y symbol can be used to test the streaming data functionality outside trading hours	D

Version	Date	Section	Description	*
2.15	2019-03-24	General	Made all the internal cross-references (e.g. "§4.1") in the User Guide linkable, for easier navigation/referencing	D
		4.1	Clarified that a reported Bid/Ask Time of 99:99:99 may indicate an invalidated quote after extended trading hours; IQFeed renamed 'Last Trade Date' field → 'Last Date'	D
		5.1 - 5.5	Clarified that IQFeed limits the historic data provided to its trial accounts vs. regular IQFeed accounts	D
		7.2	Clarified IQFeed only stores news items of past 180 days	D
		12	Split chapter 12 into two sections: *IQML* messages (12.1), IQFeed logging (12.2)	D
		12.1	Added display of a few types of IQFeed system messages (excluding the periodic stats messages) in Debug mode	F
		12.2	Added programmatic control of IQFeed logging	F
		B.2	Added new section listing just functional *IQML* changes	D
2.16	2019-04-17	5.2, 5.3	Added the ability to specify **BeginDate** for historic weekly/monthly data requests, in addition to **MaxItems**	F
2.17	2019-05-07	3.4, 5.4	Added clarifications on possible causes of run-time error; Display an explanatory message upon an IQFeed account authorization error	F
		4.1, 4.3, 5.1, 7.2, 8.1, 14	Clarified warning message in case of partial results returned due to timeout, by including a suggestion to decrease **NumOfEvents** parameter value, when relevant	F
		4.2	Clarified splits data aspects in Fundamental info reports	D
		4.4, 6.4	Added support for equity Level 2 data (in addition to futures Level 2 which was already supported); Added `ID` fields to all returned Level 2 data (futures and equities); changed default **NumOfEvents** from `inf` to 10; *IQML* now connects to IQFeed L2 servers only as needed, not in startup; Issue error message for each separate failure to access L2 data, not just during initial connection attempt; Added the **IncludeEmptyQuotes** parameter	F
		5.4	Clarified that daily/weekly/monthly interval data is available for the past 15+ years (except in IQFeed trial)	D
		5.4, 5.5	Fixed parallelization when only one of **BeginDateTime**/ **EndDateTime** is specified but not the other	F
		6.3	Clarified that there is no **EndDateTime** parameter for streaming interval bars, unlike for historic bars (§5.4)	D
		9.1	Improved behavior for users with multiple IQFeed accounts using the **Username**/**Password** parameters; Improved startup speed in case of multiple connects/ disconnects during the day	F

Version	Date	Section	Description	*
2.18	2019-05-14	1, 2.1	Clarified that on Mac/Linux, while Matlab and *IQML* can indeed run in native mode, some features (§9.5 etc.) are only available if Matlab runs under Parallels/Wine	D
		9.1	Added an informative alert when another process is using one of IQFeed's expected connection ports	F
		3.1, 9.5	Added the 'registry' action and functionality, to enable convenient access to IQFeed's registry settings	F
		12.2	Clarified that IQML('log') uses the Windows registry, so will not work if Matlab runs in Mac/Linux native mode	D
2.19	2019-07-07	Cover	Updated compatibility of supported Matlab releases	D
		-	Added an online HTML version of this User Guide (http://IQML.net/files/IQML_User_Guide)	D
		-	Added *functionSignatures.json* file for easier IQML usage in Live Editor	F
		-	Fixed compatibility issue with Matlab R2008a - R2012b	F
		3.1, 5.6	Added 'summary' action and functionality, to fetch historic end-of-day market summary (Professional license)	F
		4	Improved query responsivity in case of IQFeed error; improved performance of single-symbol queries	F
		4.1	Support for IQFeed client 6.1: new optional **Fields** (`Most Recent Trade Aggressor`, `Most Recent Trade Day Code`)	F
		4.2	Support IQFeed client 6.1: new fundamental data fields (`Session_Open_Time`, `Session_Close_Time`, `Base_Currency`, `Contract_Size`, `Contract_Months`, `Minimum_Tick_Size`, `FIGI`, `First_Delivery_Date`, and `Security_SubType`). Added a table describing all the available data fields.	F
		5.1-5.3	Included today's partial daily trading data in day/week/month history queries, in IQFeed client 6.1 or newer.	F
		5.4	Added **LabelAtBeginning** parameter for historic interval data queries (requires IQFeed client version 6 or newer)	F
		5.4	Clarified that sub-daily data may report data from non-trading days (e.g. Sunday night, when ES starts trading)	D
		6.1-6.3	Clarified that when using **ClearBuffer**, some streaming data events may be lost	D
		8.2	Default value for **IncludeBinary** parameter is now `false`	F
		9.1	Added **Protocol** parameter to customize initial connection	F
2.20	2019-07-10	4.2	Added example of new data fields in IQFeed client 6.1	D
		5.6	Added the **ReportEmptyFields**, **Filter** parameters and functionality (scanning based on multiple filter criteria)	F
		8.2	Fixed the description of the **IncludeBinary** parameter	D
		9.1	Improved IQConnect startup on Linux/Mac via wine	F

Version	Date	Section	Description	*
2.21	2019-07-14	4.6	Enabled fetching latest (current) market summary/scanner via a 'summary' query (Professional license)	F
		5.2, 5.3	Clarified the latest bar has today's data only in IQFeed 6.1+	D
		5.6	Moved most of the 'summary' query description to §4.6 and clarified the text; left only the history aspects in §5.6	D
		5.6	Default **Date** parameter value changed (yesterday→now)	F
2.22	2019-07-17	-	Fixed bug that caused a "Duplicate field name Reserved" error for some queries on some Matlab releases	F
		3.6	Added reference to §10.2 for suggested ways to speed-up callback processing overhead on data fetches	D
		4.2	Added `Underlying_Contract` field for continuous futures	F
		4.6	Fixed bug in change % fields of 'top' scanner queries; reordered 'top' results `volume` fields for better readability	F
		9.3	Added information on the client stats port fields and the importance of `KBsQueued` field for runtime performance	D
2.23	2019-09-18	-	Improved collection of parallelized results, in case some of the results returned an error or empty data	F
		4.2	Clarified that `Common_Shares_Outstanding` is in thousands	D
		4.3	Added **UseParallel** parameter feature to IntervalBars	F
		5	Fixed the implementation of **MaxItems**=-1, due to an IQFeed API behavior change	F
2.24	2019-10-05	-	Increased processing speed & streaming data throughput	F
		5.5	Added `TradeAggressor`, `DayOfMonth` tick fields (IQFeed 6.1+)	F
		8.*	Fixed duplicate lookup entries after an IQML reconnect	F
		8.1	Indicated that IQFeed has an internal bug that prevents searching for symbols with **SIC/NAICS** < 10	D
		8.5, 8.6	Indicated that IQFeed has an internal bug that does not report some SIC codes, and possibly also NAICS codes	D
		9.4	Enabled multiple custom commands in each IQML query	F
		12.2	Clarified the usage of redirecting IQFeed log file **Path**	D
2.25	2019-10-06	-	Clarified a few error messages	F
		5.*	Fixed bug in historic data introduced in release 2.24	F
		9.1	Improved detection of port conflicts with other programs	F
2.26	2019-10-26	2.1	Improved reporting of missing program components	F
		4.*	Fixed bug when multiple symbols are queried together and some symbols have no data (e.g., bad ticker)	F
		4.*	Fixed bug of not distinguishing between symbols with '.' and '-' (e.g., AGM.A, AGM-A) when both are queried	F
		4.2	Improved fundamental data query using caching	F
		4.5	Automatically infer **UnderlyingSymbol** for future option	F
		4.*, 5.*	Small performance speedup of multi-symbol queries	F

B.2 Functional change log (excluding documentation changes)

The table below is a subset of the table in §B.1, listing just functional *IQML* changes:

Version	Date	Section	Description
0.80	2017-10-17	-	Beta integration of *IQML* in a user trading program
1.00	2018-02-26	-	First commercial release of *IQML*
1.01	2018-03-11	8.1	Enabled message-specific user callbacks; Added additional information to callback eventData
1.02	2018-03-12	4.3.3	Added relevant symbols list in returned news story data
1.03	2018-03-19	3.2	Enabled **Symbol** and **Symbols** as synonymous params
		4.1, 6.1	Improved ticks request logic & the returned data fields
		6.1	Enabled requesting streaming ticks/quotes for multiple symbols at once, in a single *IQML* command
1.04	2018-04-01	2.1	Added support for native Mac IQFeed client (untested)
		3.2	Added new **MsgParsingLevel** general parameter, for improved callback run-time performance
		5	**MaxDataItems** input parameter is renamed **MaxItems**
		6.1	Some result output fields renamed for consistency; **BufferSize** input parameter is renamed **MaxItems** for consistency; clarified the documentation text
		6.2	Added new streaming regional updates functionality
		7.3	Added newline characters between separate paragraphs in the reported news-story text, for better readability
		7.5	Added new streaming news functionality
		10	Added timestamp and channel info to debug printouts
1.05	2018-04-05	3.1, 8	Added new symbols and numeric market codes lookup functionality
		3.2, 4.1 5, 7.2	Modified the default Timeout value from 3 to 5 [secs]
		8.2, 10.3	Added basic support for options-chain and futures-chain symbol lookup (better support is planned for next version)
		9.1	Enabled specifying IQFeed **Username** and **Password**; Added a 10-sec timeout on IQFeed connection attempts
		9.3	Added extra port-specific stats when **AddPortStats**=1
1.06	2018-04-08	8.2	Added options/futures chain lookup functionality
1.07	2018-04-10	9.1	Added info msgs on server connections/disconnections
1.08	2018-04-11	4.1, 6.1	Added the Symbol field to returned quotes data struct
1.09	2018-04-16	6.3	Added Interval Bars functionality
1.11	2018-05-16	4.3, 6.4, 10.5	Added Market Depth (Level 2) functionality
1.12	2018-05-23	12	Added alerts functionality

Version	Date	Section	Description
1.13	2018-05-25	7.2	Enabled auto-fetch of full news story in news headlines query (streaming/blocking) using **GetStory** parameter
		11.2	Enabled reporting the full news story (in addition to headline) in news alerts using **GetStory** parameter
		11.1, 11.2	Added regional updates alert functionality (in addition to news/quote/intervalbar alerts)
1.131	2018-05-28	3.1	Fixed bug in accepting struct-based input parameters
		7.2	Fixed bug in the news headlines functionality
1.14	2018-05-30	4.2	Enabled specifying multiple **Symbols** in a single Fundamental-data query
		6.2	Enabled specifying multiple **Symbols** in a single streaming Regional updates query
		7.3	Enabled specifying multiple news headline **ID** values in a single news story query
1.15	2018-07-08	4.1, 6.1, 14	Enabled querying snapshot (top of market) & streaming data of multiple symbols at once, in a single *IQML* query
		4.2	Fixed: querying multi-symbol fundamental data sometimes returned empty results
		6.1-6.3	Fixed: debug data was displayed when streaming queries were requested (now only displayed if **Debug**=1)
		8.2	Enabled querying fundamental data of all symbols in an options/futures chain at once, in a single *IQML* query
		8.2	Enabled querying snapshot (top of market) data of entire options/futures chain at once, in a single query
		9.1	Fixed: *IQML* query during IQFeed connection sometimes returned empty/error results
1.16	2018-07-09	5	Improved performance (speed) of historical data queries
1.17	2018-07-30	6	Enabled retrieval and cancellation of streaming data for multiple/all streamed symbols in a single *IQML* command
1.18	2018-08-03	3.1, 3.5	Added optional errorMsg output for *IQML* commands
		9.1	Fixed problem of duplicate fields during initial connection
		9.1	Improved the reliability of a programmatic IQFeed disconnect/reconnect
1.19	2018-08-06	3.2, 12	Added the **RaiseErrorMsgs** parameter to control whether IQFeed errors should raise a Matlab error
		4.1, 5.1, 7.2, 8.1	Message about partial data received due to **Timeout** is now a Matlab warning message, not an error message
1.20	2018-08-07	5, 14	Enabled requesting history data for multiple symbols in a single *IQML* command
		5.1, 5.4, 5.5	Automatically convert **BeginDate**⇔**BeginDateTime**, **EndDate**⇔**EndDateTime** (i.e. try to fix usage error)

Version	Date	Section	Description
1.21	2018-08-10	8.1	Enabled looking up symbols by market(s), sec-type(s)
1.22	2018-08-13	8.2	Enabled **NearMonths** values of 0-12, not just 0-4, for options/futures chain. Note: this is based on undocumented IQFeed functionality, so might not work in some cases.
		3.2 etc.	Limited the **Timeout** parameter values to 0-3000 [secs]
1.23	2018-08-14	9.1	Fixed a problem of possible bad connection to IQFeed during the initial connection by *IQML*
1.24	2018-08-31	3.2 etc.	Limited the **Timeout** parameter values to 0-9000 [secs], with 0 indicating infinite (i.e. no-limit) timeout
		8.2	Enabled **NearMonths** values of 0-99, not just 0-12, for options/futures chain. Note: this is based on undocumented IQFeed functionality, so might not work in some cases.
		9.1	Enabled multiple Matlab processes on the same computer to run *IQML* concurrently (Beta)
2.00 (major update)	2018-09-05		*This is a major update. Highlights: query parallelization and multiple usability/functionality fixes/improvements*
		(all)	Enabled parallel processing of *IQML* commands within `parfor`/`spmd` blocks, and parallel internal processing via the **UseParallel** parameter (Professional license only)
		2	Added the license type to the output of *IQML('version')*
		3.5	All returned data arrays are now column vectors
		3.5	Using the 2nd (optional) output of *IQML* (`errorMsgs`) now implies a default value of `false` for **RaiseErrorMsgs**
		4	Modified reported data format when **NumOfEvents** >1
		4.1, 8.2	Issued a warning when requesting more symbol quotes than your IQFeed account limit
		4.3	Added new section on blocking interval bars functionality
		4.3, 5.4, 6.3	Clarified that **IntervalSize** must be ≥100 for volume bars (a new limitation of IQFeed)
		4.3, 5, 6.3, 7.4	Enabled specifying dates and date-times using Matlab `datetime` objects (in addition to datenums and strings)
		6	Added `Symbol` field to returned streaming data struct
		7.4	Story count for symbols that have no related news story is reported as 0 (such symbols were previously skipped)
		9.3	Added `Exchanges`, `ServerVersion`, `ServiceType` fields to the returned client stats data
		11	Fixed various things with the Alerts functionality
2.02	2018-09-13	4.3, 5.4, 6.3	Clarified that **IntervalSize** must be < 86400 for secs bars (a new limitation of IQFeed); added warning when user attempts to use an invalid **IntervalSize** value.
		9.1	Added detection & report for a case of a non-communicative background IQConnect process
2.03	2018-09-30	9.1	Fixed a problem with the license check that caused IQFeed disconnections

Version	Date	Section	Description
2.04	2018-10-02	5	Improved download speed of historical data queries
		6, 7.5, 9.2	`LatestEventTimestamp` is now reported in seconds (not msecs) resolution by default, unless **Debug** is 1 or `true`
2.05	2018-10-13	4.1, 6.1	Added **Fields** parameter to enable dynamic fields-set
2.06	2018-10-15	6.1	Minor fixes, performance speedup of streaming quotes
2.07	2018-10-21	5.4	Enabled using **MaxDays** as synonym for the **Days** parameter in historic interval queries
		9.1	Fixed a problem with the license validation that prevented connection in certain cases
		9.4	Fixed a few small edge-cases with sending custom commands to IQFeed
2.08	2018-10-28	5.1, 5.4, 5.5	Enabled parallelized historic data queries (daily/interval/ticks) that have date/time range (Professional license only)
2.09	2018-11-07	3.1	Fixed a bug in parsing input parameters in struct format
2.10	2018-11-14	2.4	Added ability to revert back to the previous *IQML* version at any time.
		4.1	Added description fields for the `Bid_Market_Center`, `Ask_Market_Center` and `Last_Market_Center` fields, when reported in a quotes message from IQFeed.
		4.5	Added a new `'Greeks'` action, to calculate Greeks, fair value price and implied volatility for options (Professional license only)
2.11	2018-11-22	4.5	**DaysPerYear** parameter was renamed **AnnualFactor**; **Duration** parameter was renamed **DaysToExpiration**; `Vega`, `Rho`, `Veta`, `Ultima` are no longer divided by 100 (compatibility with Matlab Financial Toolbox, Maple & NAG); minor fix for `Veta` (negative value); Added new fields in the reported data struct: `Omega + Lambda` (synonyms), `CRho`, `Color`, `Annual_Factor_Used`. Clarified differences of *IQML*'s Greek values vs. Matlab's Trading Toolbox, NAG, and Maple. Added a table explaining all the reported Greek values.
2.12	2019-01-16	2.2	Added cross-check for *IB-Matlab* connector
		2.4	Added a new `'revert'` action, to revert back to a previous *IQML* version
		3.1	Fixed: display this User Guide using *IQML*('doc') even when the document is not on the Matlab path
2.13	2019-02-28	6.1-6.3	Added the **ClearBuffer** parameter for streaming data; fixed bug with streaming data when **NumOfEvents**=inf; fixed the documentation of **NumOfEvents** default value

Version	Date	Section	Description
2.14	2019-03-14	3.1	Fixed a problem specifying parameters via Matlab table
		3.1	Added ability to specify parameters using Matlab class object properties, similar to struct fields
		3.1, 14	Invalid parameter names are now ignored (a warning is displayed), rather than raising a Matlab error/exception
		4.1 etc.	Improved handling of multi-symbol queries
		4-7, 11.1	**Symbols**, **Fields**, and news **Sources** can now be separated by ':' or ',' (for example, 'IBM:HP' or 'IBM,HP')
2.15	2019-03-24	12.1	Added display of a few types of IQFeed system messages (excluding the periodic stats messages) in Debug mode
		12.2	Added programmatic control of IQFeed logging
2.16	2019-04-17	5.2, 5.3	Added the ability to specify **BeginDate** for historic weekly/monthly data requests, in addition to **MaxItems**
2.17	2019-05-07	3.4, 5.4	Display an explanatory message upon an IQFeed account authorization error
		4.1, 4.3, 5.1, 7.2, 8.1, 14	Clarified warning message in case of partial results returned due to timeout, by including a suggestion to decrease the **NumOfEvents** parameter value, where this is relevant
		4.4, 6.4	Added support for equity Level 2 data (in addition to futures Level 2, which was already supported); Added `ID` fields to all returned Level 2 data (futures and equities); changed default **NumOfEvents** from `inf` to 10; *IQML* now connects to IQFeed L2 servers only as needed, not in startup; Issue an error message for each separate failure to access L2 data, not just during the initial connection attempt; Added the **IncludeEmptyQuotes** parameter
		5.4, 5.5	Fixed parallelization when only one of **BeginDateTime** / **EndDateTime** is specified but not the other
		9.1	Improved behavior for users with multiple IQFeed accounts using the **Username**/**Password** parameters; Improved startup speed in case of multiple connects/ disconnects during the day
2.18	2019-05-14	9.1	Added an informative alert when another process is using one of IQFeed's expected connection ports
		3.1, 9.5	Added the 'registry' action and functionality, to enable convenient access to IQFeed's registry settings

Version	Date	Section	Description
2.19	2019-07-07	-	Added *functionSignatures.json* file for easier IQML usage in Live Editor
		-	Fixed compatibility problem with old Matlab releases (R2008a - R2012b)
		3.1, 5.6	Added the 'summary' action and functionality, to fetch historic end-of-day market summary (Professional license)
		4	Improved query responsivity in case of IQFeed error; improved performance of single-symbol queries
		4.1	Support IQFeed client 6.1: new quotes data **Fields** (`Most Recent Trade Aggressor, Most Recent Trade Day Code`)
		4.2	Support IQFeed client 6.1: new fundamental data fields (`Session_Open_Time, Session_Close_Time, Base_Currency, Contract_Size, Contract_Months, Minimum_Tick_Size, FIGI, First_Delivery_Date`, and `Security_SubType`)
		5.1-5.3	Included today's partial daily trading data in day/week/ month history queries, in IQFeed client 6.1 or newer.
		5.4	Added **LabelAtBeginning** parameter for historic interval data queries (requires IQFeed client version 6 or newer)
		8.2	Default value for **IncludeBinary** parameter is now `false`
		9.1	Added **Protocol** parameter to customize initial connection
2.20	2019-07-10	5.6	Added the **ReportEmptyFields**, **Filter** parameters and functionality (scanning based on multiple filter criteria)
		9.1	Improved IQConnect startup on Linux/Mac via wine
2.21	2019-07-14	4.6	Enabled fetching latest (current) market summary/scanner via a 'summary' query (Professional license)
		5.6	Default **Date** parameter value changed (yesterday→now)
2.22	2019-07-17	-	Fixed a bug that caused a "Duplicate field name Reserved" error for some queries on some Matlab releases
		4.2	Added `Underlying_Contract` field for continuous futures
		4.6	Fixed bug in change % fields of 'top' scanner queries; reordered 'top' results `volume` fields for improved readability
2.23	2019-09-18	-	Improved collection of parallelized results, in case some of the results returned an error or empty data
		4.3	Added **UseParallel** parameter functionality to IntervalBars
		5	Fixed the implementation of **MaxItems**=-1, due to an IQFeed API behavior change
2.24	2019-10-05	-	Improved processing speed and streaming data throughput
		5.5	Added `TradeAggressor`, `DayOfMonth` fields to reported ticks data (IQFeed client 6.1 or newer)
		8.*	Fixed duplicate lookup entries after an IQML reconnect
		9.4	Enabled multiple custom commands in single IQML query

Version	Date	Section	Description
2.25	2019-10-06	-	Clarified a few error messages
		5.*	Fixed bug in historic data that was introduced in release 2.24
		9.1	Improved detection of port conflicts with other programs
2.26	2019-10-25	2.1	Improved reporting of missing program components
		4.*	Fixed bug when multiple symbols are queried together and some symbols have no data (e.g., bad ticker)
		4.*	Fixed bug of not distinguishing between symbols with '.' and '-' (e.g., AGM.A, AGM-A) when they are both queried
		4.2	Added fundamental data caching for improved performance
		4.5	Automatically infer **UnderlyingSymbol** for future options
		4.*, 5.*	Small performance speedup of multi-symbol queries