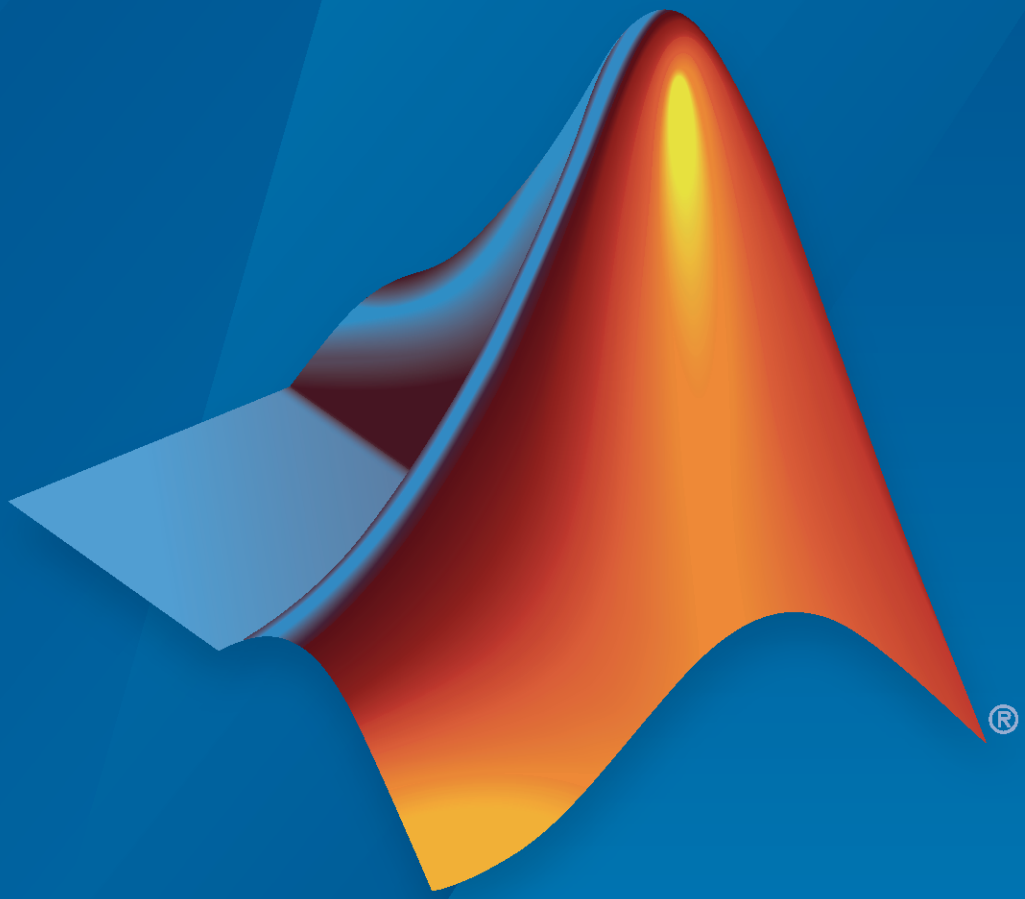
# Datafeed Toolbox™

User's Guide



MATLAB®

R2020b

MathWorks®

# How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

*Datafeed Toolbox™ User's Guide*

**Revision History**

| | | |
|---|---|---|
| December 1999 | First printing | New for MATLAB® 5.3 (Release 11) |
| June 2000 | Online only | Revised for Version 1.2 |
| December 2000 | Online only | Revised for Version 1.3 |
| February 2003 | Online only | Revised for Version 1.4 |
| June 2004 | Online only | Revised for Version 1.5 (Release 14) |
| August 2004 | Online only | Revised for Version 1.6 (Release 14+) |
| September 2005 | Second printing | Revised for Version 1.7 (Release 14SP3) |
| March 2006 | Online only | Revised for Version 1.8 (Release 2006a) |
| September 2006 | Online only | Revised for Version 1.9 (Release 2006b) |
| March 2007 | Third printing | Revised for Version 2.0 (Release 2007a) |
| September 2007 | Online only | Revised for Version 3.0 (Release 2007b) |
| March 2008 | Online only | Revised for Version 3.1 (Release 2008a) |
| October 2008 | Online only | Revised for Version 3.2 (Release 2008b) |
| March 2009 | Online only | Revised for Version 3.3 (Release 2009a) |
| September 2009 | Online only | Revised for Version 3.4 (Release 2009b) |
| March 2010 | Online only | Revised for Version 3.5 (Release 2010a) |
| September 2010 | Online only | Revised for Version 4.0 (Release 2010b) |
| April 2011 | Online only | Revised for Version 4.1 (Release 2011a) |
| September 2011 | Online only | Revised for Version 4.2 (Release 2011b) |
| March 2012 | Online only | Revised for Version 4.3 (Release 2012a) |
| September 2012 | Online only | Revised for Version 4.4 (Release 2012b) |
| March 2013 | Online only | Revised for Version 4.5 (Release 2013a) |
| September 2013 | Online only | Revised for Version 4.6 (Release 2013b) |
| March 2014 | Online only | Revised for Version 4.7 (Release 2014a) |
| October 2014 | Online only | Revised for Version 5.0 (Release 2014b) |
| March 2015 | Online only | Revised for Version 5.1 (Release 2015a) |
| September 2015 | Online only | Revised for Version 5.2 (Release 2015b) |
| March 2016 | Online only | Revised for Version 5.3 (Release 2016a) |
| September 2016 | Online only | Revised for Version 5.4 (Release 2016b) |
| March 2017 | Online only | Revised for Version 5.5 (Release 2017a) |
| September 2017 | Online only | Revised for Version 5.6 (Release 2017b) |
| March 2018 | Online only | Revised for Version 5.7 (Release 2018a) |
| September 2018 | Online only | Revised for Version 5.8 (Release 2018b) |
| March 2019 | Online only | Revised for Version 5.8.1 (Release 2019a) |
| September 2019 | Online only | Revised for Version 5.9 (Release 2019b) |
| March 2020 | Online only | Revised for Version 5.9.1 (Release 2020a) |
| September 2020 | Online only | Revised for Version 5.9.2 (Release 2020b) |

# Contents

## 1 Getting Started


**Datafeed Toolbox Product Description** . . . . . **1-2**

**Data Server Connection Requirements** . . . . . **1-3**
License Requirements . . . . . **1-3**
Proxy Information Requirements . . . . . **1-3**
Platform Requirements . . . . . **1-4**

**Installing Bloomberg and Configuring Connections** . . . . . **1-5**
Install Bloomberg Software . . . . . **1-5**
Add JAR Files to the MATLAB Java Class Path . . . . . **1-5**
Run Bloomberg Communications Server . . . . . **1-6**

**Configuring Enterprise Platform from Refinitiv Connections** . . . . . **1-8**
Configure a Reuters Connection . . . . . **1-8**
Configure an RTIC (TIC-RMDS Edition) Reuters Connection with DACS Authentication . . . . . **1-9**
Configure an RTIC (TIC-RMDS Edition) Reuters Connection Without DACS Authentication . . . . . **1-11**
Troubleshoot the Reuters Configuration Editor . . . . . **1-12**

**Retrieve Current and Historical Data Using Bloomberg** . . . . . **1-14**

**Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv** . . . . . **1-17**

**Retrieve Historical Data Using FRED** . . . . . **1-19**

**Retrieve Historical Data Using Quandl** . . . . . **1-21**

**Retrieve Historical Data Using Haver Analytics** . . . . . **1-23**

**Retrieve Intraday and Historical Data Using IQFEED** . . . . . **1-24**

**Writing and Running Custom Event Handler Functions** . . . . . **1-26**
Write a Custom Event Handler Function . . . . . **1-26**
Run a Custom Event Handler Function . . . . . **1-26**
Workflow for Custom Event Handler Function . . . . . **1-26**

## 2 Communicate with Financial Data Servers

**Communicating with Data Providers** . . . . . **2-2**

**Comparing Bloomberg Connections** . . . . . **2-3**

## 3 Data Provider Workflows

**Connect to Bloomberg** . . . . . **3-2**

**Retrieve Bloomberg Current Data** . . . . . **3-6**

**Retrieve Bloomberg Historical Data** . . . . . **3-8**

**Retrieve Bloomberg Intraday Tick Data** . . . . . **3-12**

**Retrieve Bloomberg Real-Time Data** . . . . . **3-14**

**Retrieve Data Using Bloomberg Data License** . . . . . **3-16**

**Workflow for Bloomberg** . . . . . **3-19**
Bloomberg Desktop, Bloomberg Server, or Bloomberg B-PIPE Services . . **3-19**
Bloomberg Data License Service . . . . . **3-19**

**Determine the Event Volume Indicator Using RavenPack News Analytics** . . . . . **3-21**

**Workflow for RavenPack News Analytics** . . . . . **3-24**

**Compare Player Salaries by Injury Status** . . . . . **3-25**

**Retrieve Team Standings for the Current Year** . . . . . **3-28**

## 4 Troubleshooting

**STATS.com Data Retrieval Errors** . . . . . **4-2**

## 5 Money.Net Topics

**Retrieve Current and Historical Money.Net Data** . . . . . **5-2**

**Retrieve Real-Time Money.Net Data** . . . . . **5-5**

**Retrieve Money.Net News Stories** . . . . . **5-7**

**Money.Net Error and Warning Messages** . . . . . **5-10**
Money.Net Connection Error Messages . . . . . **5-10**
Money.Net Data Retrieval Error Messages . . . . . **5-11**
Money.Net Data Retrieval Warning Messages . . . . . **5-11**

## 6 Elektron Topics

**Decide to Buy Shares Using Elektron Current Data** . . . . . **6-2**

**Decide to Buy Shares Using Elektron Real-Time Data** . . . . . **6-4**

## 7 Twitter Topics

**Conduct Sentiment Analysis Using Historical Tweets** . . . . . **7-2**

**Tweet Based on Retrieved Twitter Data** . . . . . **7-6**

## 8 Tick History from Refinitiv Topics

**Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv** . . . . . **8-2**

**Decide to Sell Shares with Historical Data Using Tick History from Refinitiv** . . . . . **8-4**

## 9 Quandl Topics

**Access Quandl Error Messages** . . . . . **9-2**

## 10 Datastream Web Services Topics

**Retrieve Datastream Web Services Historical Data** . . . . . **10-2**

**Access Datastream Web Services Error Messages** . . . . . **10-4**

## 11 IHS Markit Topics

**Retrieve Factor Rank Data for Portfolio Selection** . . . . . **11-2**

**IHS Markit Error Messages** . . . . . **11-4**

## 12 Functions

# Getting Started


- “Datafeed Toolbox Product Description” on page 1-2
- “Data Server Connection Requirements” on page 1-3
- “Installing Bloomberg and Configuring Connections” on page 1-5
- “Configuring Enterprise Platform from Refinitiv Connections” on page 1-8
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17
- “Retrieve Historical Data Using FRED” on page 1-19
- “Retrieve Historical Data Using Quandl” on page 1-21
- “Retrieve Historical Data Using Haver Analytics” on page 1-23
- “Retrieve Intraday and Historical Data Using IQFEED” on page 1-24
- “Writing and Running Custom Event Handler Functions” on page 1-26

# Datafeed Toolbox Product Description

**Access financial data from data service providers**

Datafeed Toolbox provides access to current, intraday, historical, and real-time market data from leading financial data providers. By integrating these data feeds into MATLAB®, you can perform analyses, develop models, and create visualizations that reflect current financial and market behaviors. The toolbox also provides functions to export MATLAB data to some data service providers.

You can establish connections from MATLAB to retrieve historical data or subscribe to real-time streams from data service providers. With a single function call, the toolbox lets you customize queries to access all or selected fields from multiple securities over a specified time period. You can also retrieve intraday tick data for specified intervals and store it as time series data.

Supported data providers include Bloomberg®, FactSet®, FRED®, Haver Analytics®, IQFEED®, Kx Systems®, Inc., Quandl®, SIX Financial Information, and Refinitiv™.

# Data Server Connection Requirements

| In this section... |
| --- |
| “License Requirements” on page 1-3<br>“Proxy Information Requirements” on page 1-3<br>“Platform Requirements” on page 1-4 |

## License Requirements

To connect to a data service provider, you must have a valid license for the required client software on your machine. For details, contact your data service sales representative or go to the data service provider website. For the list of websites, see “Communicating with Data Providers” on page 2-2. The following data service providers require you to install proprietary software on your computer.

- Bloomberg
  - Bloomberg Desktop, Server, or B-PIPE® software license
  - License for Bloomberg Data License
- FactSet
  - License to use FactSet DataDirect or FactSet Workstation
- Haver Analytics
- IQFEED
- Kx Systems, Inc.
- RavenPack® News Analytics
  - For details about setup and configuration, see the RavenPack Developer Zone.
- Refinitiv data servers
  - To connect from the Internet to the Datastream™ Web Services from Refinitiv API, request a user name, password, and URL from Refinitiv.

## Proxy Information Requirements

If your network requires proxy authentication, these data service providers can require specification of a proxy host, proxy port, user name, and password:

- Datastream Web Services from Refinitiv
- FactSet
- FRED
- STATS.com
- IHS Markit®
- Quandl
- Tick History from Refinitiv

For details, see “Specify Proxy Server Settings for Connecting to the Internet”.

## Platform Requirements

These data service providers work only with the Windows® platform:

- Bloomberg
- Haver Analytics
- IQFEED

## See Also

## More About

- “Communicating with Data Providers” on page 2-2
- “Installing Bloomberg and Configuring Connections” on page 1-5
- “Configuring Enterprise Platform from Refinitiv Connections” on page 1-8

# Installing Bloomberg and Configuring Connections

| In this section... |
| --- |
| “Install Bloomberg Software” on page 1-5<br>“Add JAR Files to the MATLAB Java Class Path” on page 1-5<br>“Run Bloomberg Communications Server” on page 1-6 |

Datafeed Toolbox provides various ways to connect to Bloomberg. For details, see “Comparing Bloomberg Connections” on page 2-3. Before connecting to Bloomberg, follow these required steps.

## Install Bloomberg Software

For the latest Bloomberg software, see `https://www.bloomberg.com`.

After installing the Bloomberg software, add Java® archive (JAR) files to the MATLAB Java class path and run the Bloomberg Communications Server according to these requirements.

| Step | Bloomberg Desktop | Bloomberg Server | Bloomberg B-PIPE | Bloomberg Data License |
| --- | --- | --- | --- | --- |
| Add `blpapi3.jar` JAR file to the MATLAB Java class path | Required for connection | Required for connection | Required for connection | Not required |
| Add `bbdl.jar` JAR file to the MATLAB Java class path | Not required | Not required | Not required | Required for connection |
| Add `bbdlapi.jar` JAR file to the MATLAB Java class path | Not required | Not required | Not required | Required for connection |
| Add `bbdlftp.jar` JAR file to the MATLAB Java class path | Not required | Not required | Not required | Required for connection |
| Run the Bloomberg Communications Server | Required for connection | Required for connection | Not required | Not required |

## Add JAR Files to the MATLAB Java Class Path

#### Bloomberg Desktop, Bloomberg Server, and Bloomberg B-PIPE

With the Bloomberg V3 release, install the JAR file `blpapi3.jar` from Bloomberg. This JAR file ensures that `blp`, `blpsrv`, `bpipe`, and other Bloomberg commands work correctly.

If you have already downloaded `blpapi3.jar`, find it in a folder such as `c:\blp\DAPI \blpapi3.jar` or search for it in the Bloomberg installation folder `c:\blp`. When you have found `blpapi3.jar`, go to step 3.

**Note** For each Bloomberg connection, the folder location of the downloaded JAR file can be different. For questions, contact Bloomberg.

If you have not downloaded `blpapi3.jar` from Bloomberg, download it as follows:

1. In the Bloomberg terminal, type `WAPI <GO>` to open the API Developer's Help Site screen.
2. Click API Download Center. Download and install the appropriate software.
3. Once you have `blpapi3.jar` on the system, add it to the MATLAB Java class path. There are two ways to add the JAR file:

   - Add the JAR file to the MATLAB Java class path for every MATLAB session using `javaaddpath`.

     ```
     javaaddpath c:\blp\DAPI\blpapi3.jar
     ```

     The JAR file path varies depending on the installed Bloomberg software.
   - Or, to automate adding this file, add `javaaddpath` to the `startup.m` file or add the full path for the JAR file to the `javaclasspath.txt` file.

   To decide which way is best for you, see "Startup Options in MATLAB Startup File" and "Static Path".
4. If you modify `startup.m` or `javaclasspath.txt` files, restart MATLAB.

#### Bloomberg Data License

Add the JAR file `bbdlapi.jar` to the MATLAB Java class path using the preceding JAR file instructions. For details, see the Data License Java SE API folder. Find this folder by entering `DLSD <GO>` in the Bloomberg terminal.

#### Troubleshooting Adding JAR Files

If the JAR files are missing from the MATLAB Java class path, one of these error messages display:

- Please verify that blpapi3.jar is included on MATLAB javaclasspath.
- Please verify that bbdlapi.jar is included on MATLAB javaclasspath.

To add the JAR files to the MATLAB Java class path, follow the preceding instructions.

For issues with downloading and installing JAR files, contact Bloomberg.

## Run Bloomberg Communications Server

The Bloomberg Communications Server is a program named `bbcomm.exe`. You can find this program in a folder such as `c:\blp\DAPI\bbcomm.exe` or search for it in the Bloomberg installation folder `c:\blp`. If `bbcomm.exe` does not start automatically, double-click it. An MS-DOS® command window opens and stays open while `bbcomm.exe` is running.

To avoid manually starting `bbcomm.exe` each time you turn on the computer, add `bbcomm.exe` to the startup folder:

1. Create a Windows shortcut to `bbcomm.exe`.

2 Move this shortcut to the startup folder. An example startup folder name is `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup`.

When you are finished with the Bloomberg connection, stop `bbcomm.exe` by running `bbstop.exe`. You can find `bbstop.exe` in the same folder as `bbcomm.exe`. Double-click `bbstop.exe`. The MS-DOS command window closes.

## See Also

`bdl` | `blp` | `blpsrv` | `bpipe`

## Related Examples

- “Connect to Bloomberg” on page 3-2

## More About

- “Data Server Connection Requirements” on page 1-3
- “Communicating with Data Providers” on page 2-2
- “Workflow for Bloomberg” on page 3-19

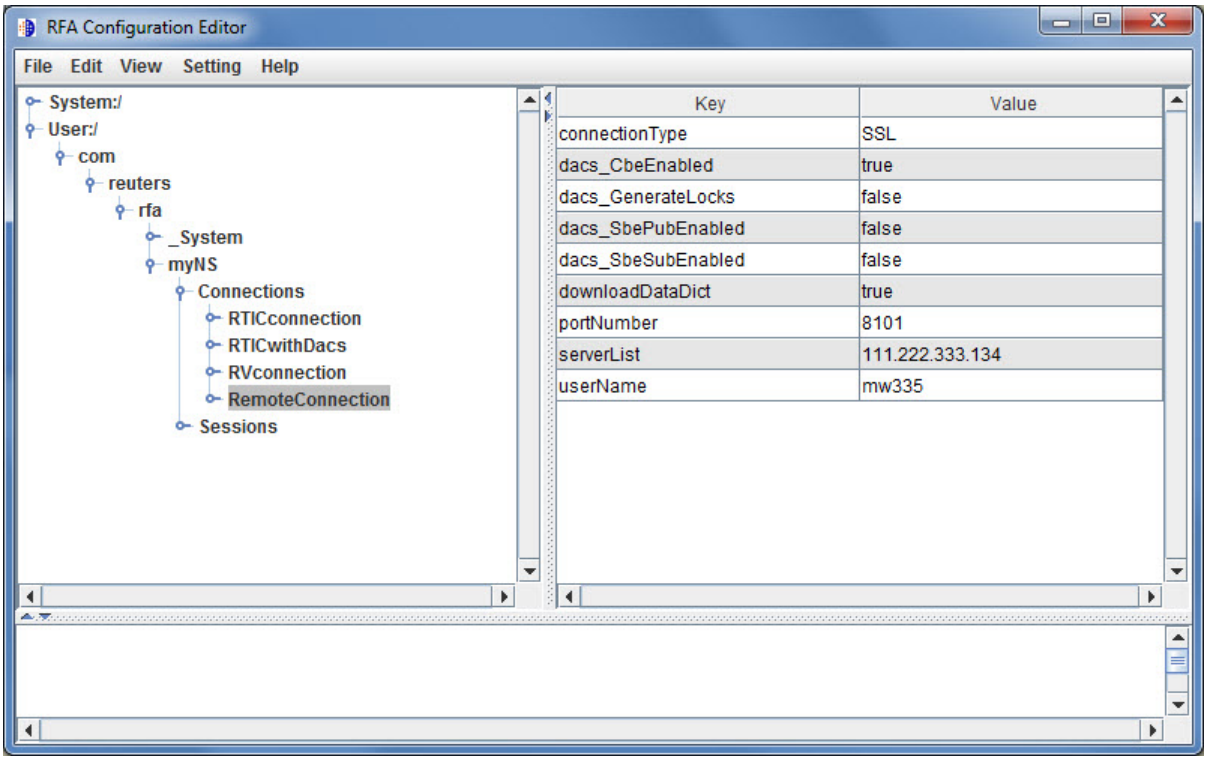
# Configuring Enterprise Platform from Refinitiv Connections

| In this section... |
| --- |
| “Configure a Reuters Connection” on page 1-8<br>“Configure an RTIC (TIC-RMDS Edition) Reuters Connection with DACS Authentication” on page 1-9<br>“Configure an RTIC (TIC-RMDS Edition) Reuters Connection Without DACS Authentication” on page 1-11<br>“Troubleshoot the Reuters Configuration Editor” on page 1-12 |

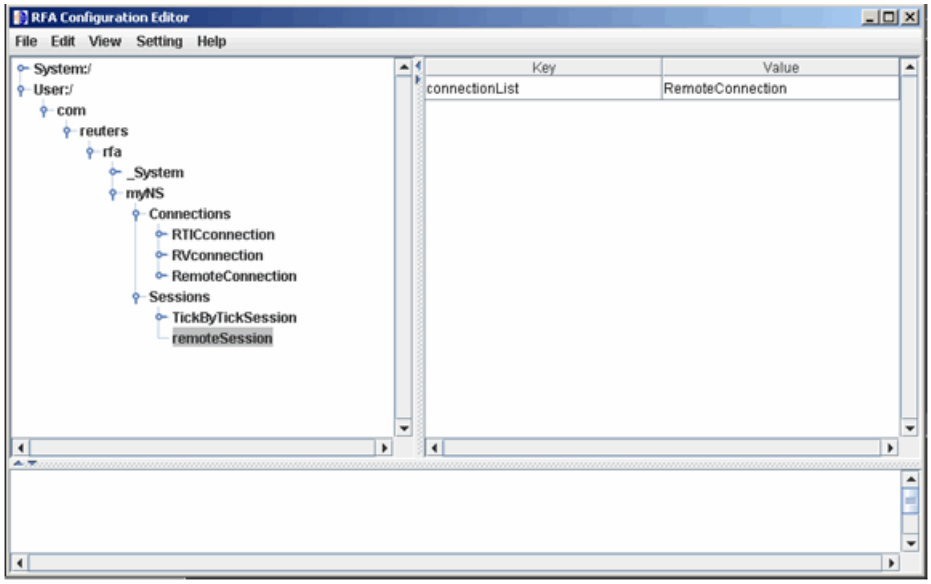
To connect to the Enterprise Platform from Refinitiv for the first time or change the authentication type, configure the Reuters® connection using the RFA Configuration Editor.

## Configure a Reuters Connection

1. Open the RFA Configuration Editor using `rmdsconfig`.
2. Load the sample configuration file by selecting **File** > **Import** > **File**. Select the file *`matlabroot`*`\toolbox\datafeed\datafeed\sampleconfig.xml`.
3. Modify `sampleconfig.xml` based on site-specific settings obtained from Reuters.
4. Define a namespace, connection, and session associated with the connection `RemoteConnection`. Set the key and value fields as shown in the RFA Configuration Editor.




This example adds the session `remoteSession` with the namespace `MyNS` to the connection list for the connection `RemoteConnection`.



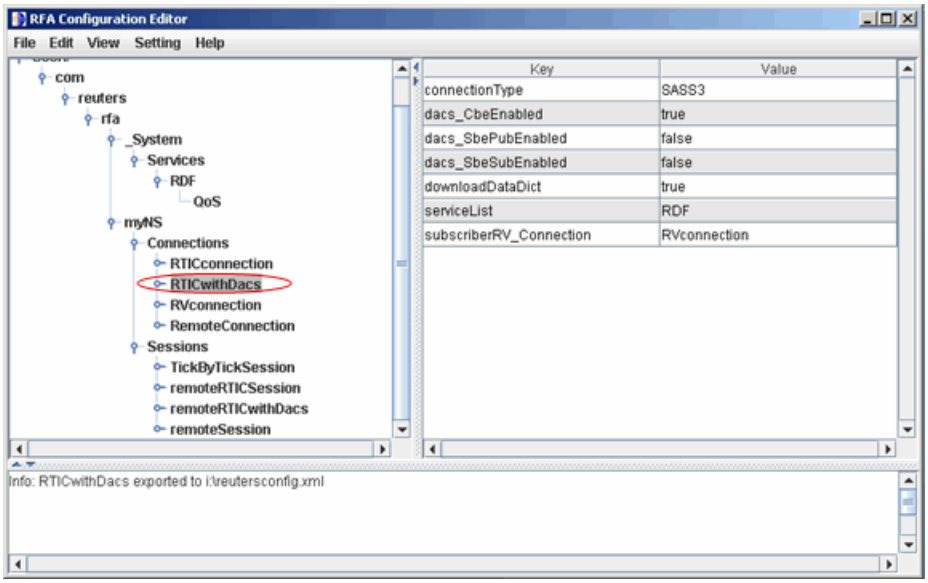

5 To connect without Data Access Control System (DACS) authentication, disable DACS by setting the keys in `RemoteConnection` to the values as shown in this table.

| Key | Value |
|---|---|
| `dacs_CbeEnabled` | `false` |
| `dacs_SbePubEnabled` | `false` |
| `dacs_SbeSubEnabled` | `false` |

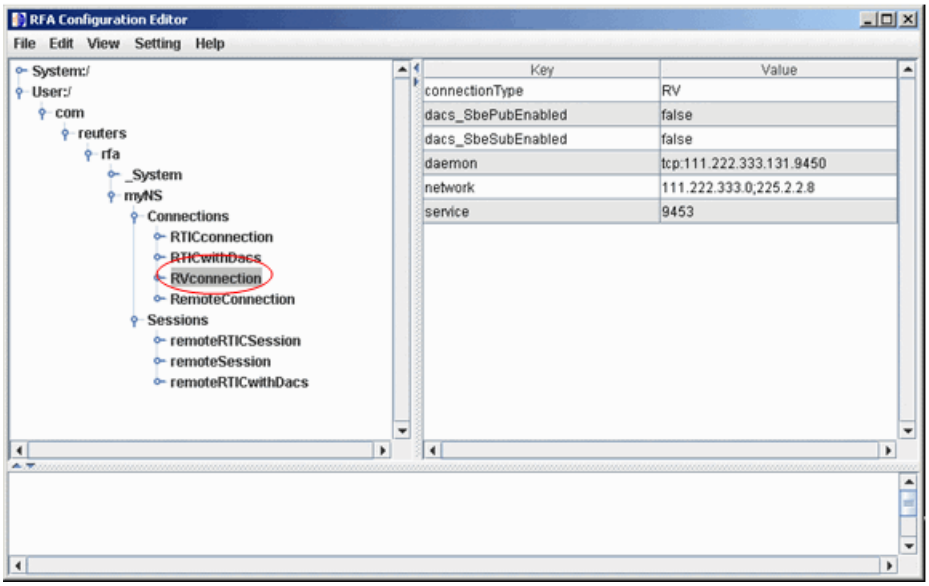
6 To run an SSL connection, set the key `dacs_GenerateLocks` to the value `false` in `RemoteConnection`.

## Configure an RTIC (TIC-RMDS Edition) Reuters Connection with DACS Authentication

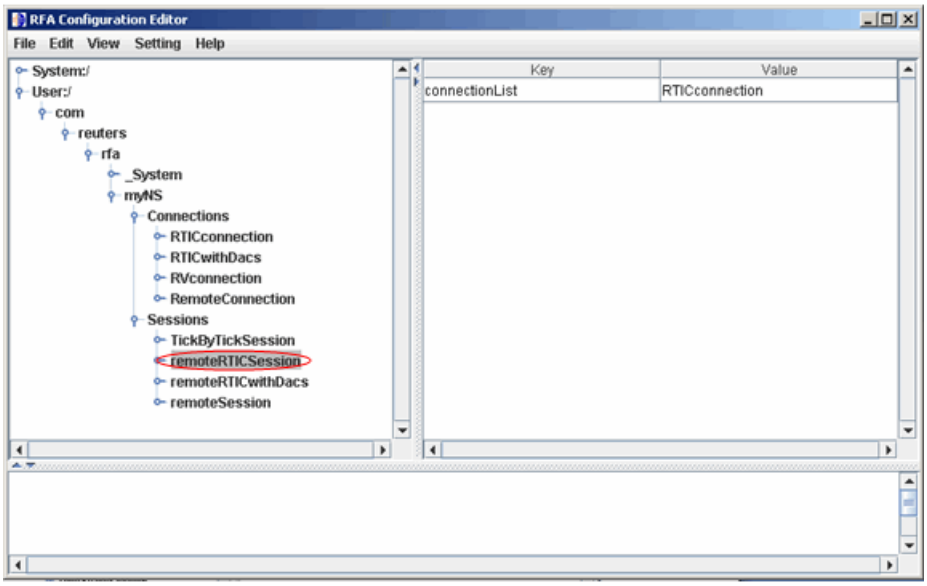
1 After loading and modifying the sample configuration file, set the keys and value fields as shown in the RFA Configuration Editor for the connection `RTICwithDacs`.




2 When you select `RVConnection`, the RTIC connection depends on the key subscriber fields shown. Set these key and value fields as shown.

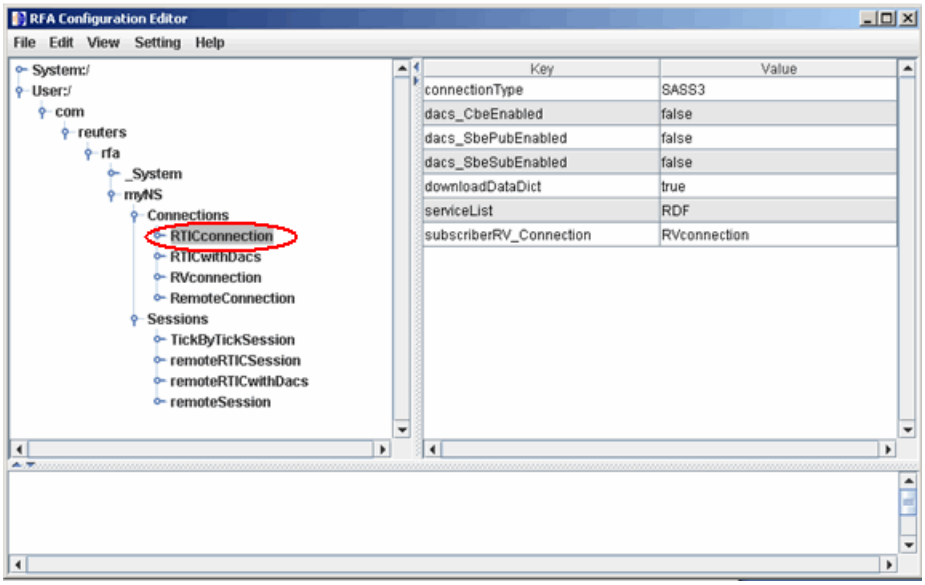



The RFA Configuration Editor shows the session `remoteRTICSession` referencing the `RTICConnection`.

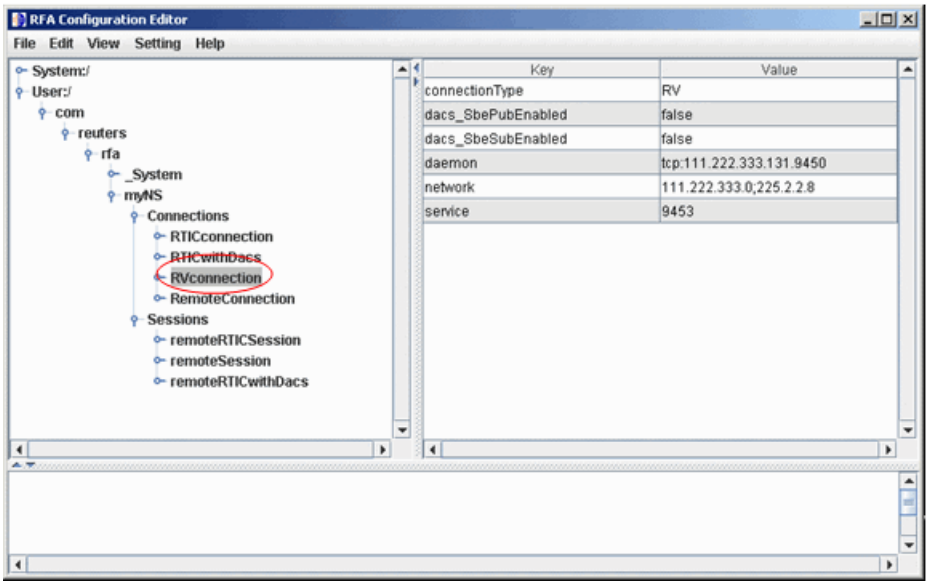



## Configure an RTIC (TIC-RMDS Edition) Reuters Connection Without DACS Authentication

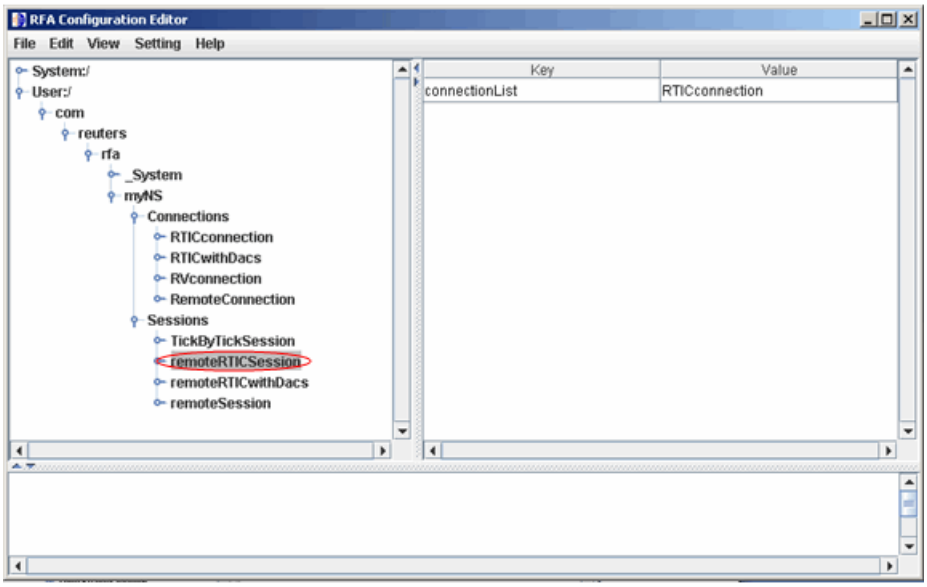
1 After loading and modifying the sample configuration file, set the key and value fields as shown for `RTICConnection`.




2 When you select `RVConnection`, the RTIC connection depends on the key subscriber fields shown. Set these key and value fields as shown.




The RFA Configuration Editor shows the session `remoteRTICSession` referencing the `RTICConnection`.




## Troubleshoot the Reuters Configuration Editor

- When you use the Reuters Configuration Editor to configure connections on a machine that does not have an XML Parser installed, these errors occur:

```
java com.reuters.rfa.tools.config.editor.ConfigEditor
org.xml.sax.SAXException: System property
org.xml.sax.driver not specified
at org.xml.sax.helpers.XMLReaderFactory.createXMLReader(Unknown
Source)
at com.reuters.rfa.tools.config.editor.rfaConfigRuleDB.rfaConfi
```

```
gRuleDB.java:56)
at com.reuters.rfa.tools.config.editor.ConfigEditor.init
(ConfigEditor.java:86)
at (com.reuters.rfa.tools.config.editor.ConfigEditor.
(ConfigEditor.java:61) at
com.reuters.rfa.tools.config.editor.ConfigEditor.main
(ConfigEditor.java:1303)
```

To address this issue, download an XML parser file, and include a path to this file in the `CLASSPATH` environment variable.

This example shows how to set the `CLASSPATH` environment variable to include the XML parser file `C:\xerces.jar` (available at `https://xerces.apache.org/xerces-j/index.html`):

```
set CLASSPATH=%CLASSPATH%;...
        matlabroot\toolbox\datafeed\datafeed\config_editor.jar;...
        c:\xerces.jar
```

- If these messages or similar messages appear in the Command Window when you establish a connection with DACS authentication:

  ```
  SEVERE: com.reuters.rfa.entitlements._Default.Global
  DACS initialization failed:
  com.reuters.rfa.dacs.AuthorizationException:
  Cannot start the DACS Library thread due to -
  Cannot locate JNI library - RFADacsLib
  ```

  then add an entry to the `$MATLAB/toolbox/local/librarypath.txt` file that points to the folder containing these files:

  - `FDacsLib.dll`
  - `sass3j.dll`
  - `sipc32.dll`

## See Also

`reuters`

## More About

- “Data Server Connection Requirements” on page 1-3
- “Communicating with Data Providers” on page 2-2

# Retrieve Current and Historical Data Using Bloomberg

This example shows how to connect to Bloomberg® and retrieve current and historical Bloomberg® market data. For details about Bloomberg® connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing a connection function. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

### Connect to Bloomberg®

Create a Bloomberg® Desktop connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg® Server using `blpsrv` or Bloomberg® B-PIPE® using `bpipe`.

### Retrieve Current Data

Format MATLAB® data display for currency.

```
format bank
```

Retrieve closing and open prices for Microsoft®.

```
sec = 'MSFT US Equity';
fields = {'LAST_PRICE';'OPEN'}; % closing and open prices

[d,sec] = getdata(c,sec,fields)
```

```
d =

  struct with fields:

    LAST_PRICE: 62.32
          OPEN: 62.48


sec =

  cell

    'MSFT US Equity'
```

`d` contains the Bloomberg® closing and open prices. `sec` contains the Bloomberg® security name for Microsoft®.

### Retrieve Historical Data

Retrieve monthly closing and open price data from January 1, 2012 through December 31, 2012 for Microsoft®.

```
fromdate = '1/01/2012'; % beginning of date range for historical data
todate = '12/31/2012'; % ending of date range for historical data
period = 'monthly'; % retrieve monthly data
```

```
[d,sec] = history(c,sec,fields,fromdate,todate,period)

d =

     734899.00         29.53         26.55
     734928.00         31.74         29.79
     734959.00         32.26         31.93
     734989.00         32.02         32.22
     735020.00         29.19         32.05
     735050.00         30.59         28.76
     735081.00         29.47         30.62
     735112.00         30.82         29.59
     735142.00         29.76         30.45
     735173.00         28.54         29.81
     735203.00         26.61         28.84
     735234.00         26.71         26.78


sec =

  cell

    'MSFT US Equity'
```

`d` contains the numeric representation of the date in the first column, closing price in the second column, and open price in the third column. Each row represents data for one month in the date range. `sec` contains the Bloomberg® security name for Microsoft®.

#### Find Maximum Open Price in Date Range

Calculate the maximum open price for the year 2012.

```
openprices = d(:,3); % retrieve all open prices in date range
max(openprices) % calculate maximum open price

ans =

         32.22
```

#### Close Bloomberg® Connection

```
close(c)
```

## See Also

`blp` | `close` | `getdata` | `history`

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8

- “Retrieve Bloomberg Intraday Tick Data” on page 3-12
- “Retrieve Bloomberg Real-Time Data” on page 3-14

# Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv

This example shows how to connect to the Enterprise Platform from Refinitiv and retrieve current and historical market data. Before connecting to Refinitiv, configure the Enterprise Platform connections. For details, see “Configuring Enterprise Platform from Refinitiv Connections” on page 1-8.

### Connect to Refinitiv

Connect to Refinitiv using a delayed connection specified by `'dIDN_RDF'`. This connection type lets you retrieve current data.

```
c = reuters('myNS::remoteSession','dIDN_RDF');
```

### Retrieve Current Data

Retrieve current data for Google®.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =

    PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains a large number of Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`. `sec` contains the Refinitiv security name for Google.

Close the Refinitiv connection.

```
close(c)
```

### Retrieve Historical Data

Connect to Refinitiv using a connection that is not delayed as specified by `'IDN_RDF'`. This connection type lets you retrieve historical data.

```
c = reuters('myNS::remoteSession','IDN_RDF');
```

Retrieve monthly market data from June 1, 2012 through December 31, 2012 for Google.

```
fromdate = '06/01/2012'; % beginning of date range for historical data
todate = '12/31/2012'; % ending of date range for historical data
period = 'm'; % monthly period for data

d = history(c,sec,fromdate,todate,period)
```

```
d =

          DATE: [7x1 double]
```

```
        CLOSE: [7x1 double]
         OPEN: [7x1 double]
         HIGH: [7x1 double]
          LOW: [7x1 double]
       VOLUME: [7x1 double]
         VWAP: [7x1 double]
    BLOCK_VOL: [7x1 double]
          ASK: [7x1 double]
          BID: [7x1 double]
```

`d` is a structure with the following fields:

- Date
- Closing price
- Open price
- High price
- Low price
- Volume
- Volume weighted average price (VWAP)
- Block volume
- Ask price
- Bid price

For this example, the structure fields contain market data from June through December.

Display the open price.

```
d.OPEN
```

```
ans =

        702.24
        679.50
        759.05
        ...
```

**Close the Refinitiv Connection**

```
close(c)
```

## See Also

`close` | `fetch` | `history` | `reuters`

# Retrieve Historical Data Using FRED

This example shows how to connect to FRED®, retrieve historical foreign exchange rates, and determine when the highest rate occurred.

### Create FRED Connection

Connect to the FRED data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

### Retrieve Historical Foreign Exchange Rates

Adjust the display data format for currency.

```
format bank
```

Retrieve all historical data for the US / Euro Foreign Exchange Rate series. `d` contains the series description.

```
series = 'DEXUSEU';
```

```
d = fetch(c,series)
```

```
d = struct with fields:
                 Title: ' U.S. / Euro Foreign Exchange Rate'
              SeriesID: ' DEXUSEU'
                Source: ' Board of Governors of the Federal Reserve System (US)'
               Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
             Frequency: ' Daily'
                 Units: ' U.S. Dollars to One Euro'
             DateRange: ' 1999-01-04 to 2018-06-15'
           LastUpdated: ' 2018-06-18 3:51 PM CDT'
                 Notes: ' Noon buying rates in New York City for cable transfers payable in fore
                  Data: [5075×2 double]
```

Display the numeric representation of the date and the foreign exchange rate for the first three rows of data.

```
d.Data(1:3,:)
```

```
ans = 3×2

     730124.00          1.18
     730125.00          1.18
     730126.00          1.16
```

### Retrieve Historical Foreign Exchange Rates Using Date Range

Retrieve historical data from January 1 through June 1, 2012, for the US / Euro Foreign Exchange Rate series.

```
startdate = '01/01/2012'; % beginning of date range for historical data
enddate = '06/01/2012'; % ending of date range for historical data
```

```
d = fetch(c,series,startdate,enddate)
```

```
d = struct with fields:
                 Title: ' U.S. / Euro Foreign Exchange Rate'
              SeriesID: ' DEXUSEU'
                Source: ' Board of Governors of the Federal Reserve System (US)'
               Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
             Frequency: ' Daily'
                 Units: ' U.S. Dollars to One Euro'
             DateRange: ' 1999-01-04 to 2018-06-15'
           LastUpdated: ' 2018-06-18 3:51 PM CDT'
                 Notes: ' Noon buying rates in New York City for cable transfers payable in forei
                  Data: [110×2 double]
```

#### Determine Highest Foreign Exchange Rate in Date Range

Determine the highest foreign exchange rate `maxforex` in the date range. `forex` contains all the exchange rates in the data.

```
forex = d.Data(:,2);
maxforex = max(forex)
```

```
maxforex =
          1.35
```

Determine when the highest foreign exchange rate occurred. To find the index `idx` for the highest foreign exchange rate, the function `find` uses the tolerance `value`. Retrieve the serial date number by indexing into the array of data using `idx`. Convert the serial date number to a character vector using the `datestr` function.

```
value = abs(forex-maxforex);
idx = find(value<0.001,1);
date = d.Data(idx,1);
datestr(date)
```

```
ans =
'24-Feb-2012'
```

#### Close FRED Connection

```
close(c)
```

## See Also

`close` | `datestr` | `fetch` | `find` | `fred`

# Retrieve Historical Data Using Quandl

This example shows how to retrieve historical data with an applied calculation from Quandl. Also, the example limits and sorts the rows in the returned data. The example assumes that you have obtained an API key from Quandl.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security from January 1, 2017 through December 31, 2017. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. Apply these conditions to the returned data:

- Use a monthly periodicity for the returned data.
- Transform the historical price data by calculating the row-on-row percent change.
- Specify the calculation using the name-value argument `"transform"` with the `"rdiff"` value.
- Limit the returned data to the first six rows using the name-value argument `"limit"`.
- Sort the dates in ascending order using the name-value argument `"order"`.

`d` is a timetable with the time in the first variable and the percent change of the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
startdate = datetime('01-01-2017','InputFormat','MM-dd-yyyy');
enddate = datetime('12-31-2017','InputFormat','MM-dd-yyyy');
periodicity = 'monthly';
d = history(c,s,startdate,enddate,periodicity, ...
    "transform","rdiff","limit",6,"order","asc");
```

Display the sorted percent changes.

```
d
```

```
d =

  6×1 timetable

       Time        PreviousSettlement
    ___________    __________________

    31-Mar-2017          -0.01
    30-Apr-2017           0.02
    31-May-2017           0.02
    30-Jun-2017           0.21
    31-Jul-2017          -0.01
    31-Aug-2017          -0.08
```

Decide to buy or sell this contract based on the historical percent changes.

## See Also

`history` | `quandl`

## More About

- “Access Quandl Error Messages” on page 9-2

## External Websites

- Quandl

# Retrieve Historical Data Using Haver Analytics

This example shows how to connect to Haver Analytics and retrieve historical data.

### Connect to Haver Analytics

Connect to Haver Analytics using a daily file.

```
c = haver('c:\work\haver\haverd.dat');
```

### Retrieve All Historical Data

Retrieve all historical data for the Haver Analytics variable FFED. The descriptor for this variable is Federal Funds [Effective] Rate (% p.a.).

```
variable = 'FFED'; % return data for FFED

d = fetch(c,variable);
```

Display the first three rows of data.

```
d(1:3,:)

ans =

     715511.00          2.38
     715512.00          2.50
     715515.00          2.50
```

d contains the numeric representation of the date and the closing value.

### Retrieve Historical Data for a Date Range

Retrieve historical data from January 1, 2005 through December 31, 2005 for FFED.

```
fromdate = '01/01/2005'; % beginning of date range for historical data
todate = '12/31/2005'; % ending of date range for historical data

d = fetch(c,variable,fromdate,todate);
```

### Close the Haver Analytics Connection

```
close(c)
```

### Open the Haver Analytics User Interface

Use the `havertool` function to open the Haver Analytics User Interface. You can observe different Haver Analytics variables in a chart format.

```
c = haver('c:\work\haver\haverd.dat');

havertool(c)
```

For details, see the `havertool` function.

## See Also

`close` | `fetch` | `haver` | `havertool`

# Retrieve Intraday and Historical Data Using IQFEED

This example shows how to connect to IQFEED and retrieve intraday and historical data. To run this example, you must first install the IQFEED Client. To download the software, see Download IQFEED Client.

### Connect to IQFEED

The following code assumes you are connecting to IQFEED using the user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

### Retrieve Intraday Data

Retrieve today's intraday data for IBM®.

```
sec = 'IBM';
fromdate = now-0.05; % beginning of date range for intraday data
                     % (approximately one hour ago)
todate = now; % ending of date range for intraday data (current time today)

timeseries(c,sec,{fromdate,todate})
```

`timeseries` creates the workspace variable `IQFeedTimeseriesData` and populates it with the intraday data. `sec` contains the IQFEED security name for IBM.

Display the first three rows of intraday data.

```
IQFeedTimeseriesData(1:3,:)
```

```
ans =

    '2013-12-19 10:09:15'    '179.5750'    '100'     '1155752'    '179.5700'    '179.6100'    '219184'    '0'    '0'    'C'
    '2013-12-19 10:09:15'    '179.5700'    '100'     '1155652'    '179.5700'    '179.6100'    '219177'    '0'    '0'    'C'
    '2013-12-19 10:09:15'    '179.5844'    '1345'    '1155552'    '179.5700'    '179.6100'    '219176'    '0'    '0'    'C'
```

The columns in `IQFeedTimeseriesData` are:

- Timestamp.
- Last price.
- Last size.
- Total volume.
- Bid price.
- Ask price.
- Tick identifier.
- The last column is the basis for last trade.

The remaining two columns are reserved for later use by the IQFEED API.

Close the IQFEED connection.

```
close(c)
```

### Retrieve Historical Data

Connect to IQFEED.

```
c = iqf('username','pwd');
```

Retrieve the last five weeks of historical data for IBM.

```
interval = 5; % number of weeks to return data
period = 'Weekly'; % retrieve weekly data

history(c,sec,interval,period)
```

`history` creates the workspace variable `IQFeedHistoryData` and populates it with the historical data.

Display the first three rows of historical weekly data.

```
IQFeedHistoryData(1:3,:)
```

```
ans =

    '2013-12-18 10:11:32'    '178.7400'    '172.7300'    '173.2200'    '178.7000'    '18695843'    '0'
    '2013-12-13 10:11:32'    '178.1520'    '172.7300'    '177.9900'    '172.8000'    '21871929'    '0'
    '2013-12-06 10:11:32'    '179.5900'    '175.1600'    '179.4600'    '177.6700'    '24819146'    '0'
```

Each row of data represents the last day of a week. The first row contains data for the last business day in the current week. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

### Close the IQFEED Connection

```
close(c)
```

## See Also

`close` | `history` | `iqf` | `timeseries`

# Writing and Running Custom Event Handler Functions

| In this section... |
| --- |
| “Write a Custom Event Handler Function” on page 1-26 |
| “Run a Custom Event Handler Function” on page 1-26 |
| “Workflow for Custom Event Handler Function” on page 1-26 |

## Write a Custom Event Handler Function

You can process events related to any data updates by writing a custom event handler function for use with Datafeed Toolbox. For example, you can monitor prices before creating an order or plot interval data in a graph. Follow these basic steps to write a custom event handler.

1. Choose the events you want to process, monitor, or evaluate.
2. Decide how the custom event handler processes these events.
3. Determine the input and output arguments for the custom event handler function.
4. Write the code for the custom event handler function.

For details, see “Create Functions in Files”. For a code example of a Bloomberg event handler function, enter `edit v3stockticker.m` at the command line.

## Run a Custom Event Handler Function

You can run the custom event handler function by passing the function name as an input argument into an existing function. For the Enterprise Platform from Refinitiv function `fetch`, specify the custom event handler as a character vector or string. For other functions, specify the custom event handler function name either as a character vector, string, or function handle. For details about function handles, see “Create Function Handle”.

For example, suppose you want to retrieve real-time data from Bloomberg using `realtime` with the custom event handler function named `eventhandler`. You can use either of these syntaxes to run `eventhandler`. This code assumes a Bloomberg connection `c`, security list `s`, Bloomberg data fields `f`, Bloomberg subscription `subs`, and MATLAB timer `t`.

Use a character vector or string.

```
[subs,t] = realtime(c,s,f,'eventhandler');
```

Or, use a function handle.

```
[subs,t] = realtime(c,s,f,@eventhandler);
```

## Workflow for Custom Event Handler Function

This workflow summarizes the basic steps to work with a custom event handler function for any of the data service providers.

1. Write a custom event handler function and save it to a file.
2. Create a connection to the data service provider.

3. Subscribe to a specific security using an existing function or API syntax.
4. Run an existing function to receive data updates and use the custom event handler function as an input argument.
5. Stop data updates by using `stop` or closing the connection to the data service provider.
6. Close the connection to the data service provider if the connection is still open.

## See Also

`fetch` | `realtime`

## More About

- “Create Functions in Files”
- “Create Function Handle”

# Communicate with Financial Data Servers


- “Communicating with Data Providers” on page 2-2
- “Comparing Bloomberg Connections” on page 2-3

# Communicating with Data Providers

Datafeed Toolbox supports connection to the data providers listed in this table. The table shows the connection functions for each data provider.

| Data Provider | Website | Connection Function | API Documentation |
|---|---|---|---|
| Bloomberg | `bloomberg.com` | `blp`, `blpsrv`, `bpipe`, or `bdl` | *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal |
| FactSet | `factset.com` | `factset` or `fds` | Not available |
| FRED | `https://research.stlouisfed.org/fred2` | `fred` | Not available |
| Haver Analytics | `haver.com` | `haver` | Not available |
| IHS Markit | `ihsmarkit.com` | `ihsmarkitrs` | IHS Markit Research Signals REST Documentation |
| IQFEED | `iqfeed.net` | `iqf` | IQFEED API |
| Kx Systems, Inc. | `kx.com` | `kx` | Kx Systems, Inc., GitHub Repository |
| Money.Net | `money.net` | `moneynet` | Money.Net API Documentation |
| Quandl | `quandl.com` | `quandl` | Quandl Documentation |
| RavenPack News Analytics | `ravenpack.com` | `ravenpack` | RavenPack Developer Zone Overview |
| SIX Financial Information | `six-financial-information.com` | `tlkrs` | Not available |
| STATS.com | `stats.com` | `statsllc` | Not available |
| Refinitiv | `refinitiv.com` | `datastreamws`, `reuters`, or `trth` | Not available |

## See Also

## More About

- "Data Server Connection Requirements" on page 1-3

# Comparing Bloomberg Connections

Datafeed Toolbox uses these different Bloomberg services to connect to Bloomberg. To learn about the connection functions and the data access functionality of each service, see this table.

You need a valid Bloomberg license to work with each Bloomberg service.

| Bloomberg Service | Bloomberg Desktop | Bloomberg Server | Bloomberg B-PIPE | Bloomberg Data License |
|---|---|---|---|---|
| Connection function | `blp` | `blpsrv` | `bpipe` | `bdl` |
| Data access | Applications obtain data from the Bloomberg Data Center by connecting locally to the Bloomberg Communications Server | Applications obtain data from the Bloomberg Data Center using a dedicated process that optimizes network resources | Provides entitled users access to permission data from the Bloomberg Data Center through the Bloomberg Appliance | Provides access to Bloomberg data using custom request files |

Each connection function has a different syntax for creating a Bloomberg connection. The connection objects created by running these functions have different properties. For details, see the respective function reference page.

For details about these services, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

For details about Bloomberg Data License, see the relevant guides by entering `DLSD` and clicking **<GO>** in the Bloomberg terminal.

## See Also

## Related Examples

- "Connect to Bloomberg" on page 3-2

## More About

- "Installing Bloomberg and Configuring Connections" on page 1-5
- "Data Server Connection Requirements" on page 1-3

# Data Provider Workflows

# Connect to Bloomberg

This example shows how to create a connection to Bloomberg using these Bloomberg services: Bloomberg Desktop, Bloomberg Server, B-PIPE, and Bloomberg Data License. For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing a connection function. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

### Create the Bloomberg Desktop Connection

```
c = blp
```

```
c =
  blp with properties:
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: 'localhost'
         port: 8194
      timeout: 0
```

`blp` creates a Bloomberg connection object `c` and returns its properties.

Validate the connection `c`.

```
v = isconnection(c)
```

```
v =

     1
```

`v` returns `true` showing that the Bloomberg connection is valid.

Retrieve the Bloomberg Desktop connection properties.

```
v = get(c)
```

```
v =

      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: 'localhost'
         port: 8194
      timeout: 0
```

`v` is a structure containing the Bloomberg session object, IP address, port number, and timeout value.

Close the Bloomberg Desktop connection.

```
close(c)
```

### Create the Bloomberg Server Connection

Connect to the Bloomberg Server using the IP addresses of the machine running the Bloomberg Server. This code assumes the following:

- The Bloomberg UUID is `12345678`.
- The IP address `serverip` for the machine running the Bloomberg Server is `'111.11.11.111'`.

```
uuid = 12345678;
serverip = '111.11.11.111';
```

```
c = blpsrv(uuid,serverip)

c =

  blpsrv with properties:

         uuid: 12345678
         user: [1x1 com.bloomberglp.blpapi.impl.aT]
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: '111.11.11.111'
         port: 8195
      timeout: 0
```

`blpsrv` connects to the machine running the Bloomberg Server on the default port number `8195`. `blpsrv` creates the Bloomberg Server connection object `c`.

Close the Bloomberg Server connection.

```
close(c)
```

### Create the B-PIPE Connection

Create a Bloomberg B-PIPE connection using the IP address of the machine running the Bloomberg B-PIPE process. This code assumes the following:

- The authentication is Windows Authentication by setting `authorizationtype` to `'OS_LOGON'`.
- The application name is blank because you are not connecting to Bloomberg B-PIPE using an application.
- The IP address `serverip` for the machine running the Bloomberg B-PIPE process is `'111.11.11.112'`.
- The port number is `8194`.

```
authorizationtype = 'OS_LOGON';
applicationname = '';
serverip = {'111.11.11.112'};
portnumber = 8194;

c = bpipe(authorizationtype,applicationname,serverip,portnumber)

c =

  bpipe with properties:

    appauthtype: ''
       authtype: 'OS_LOGON'
        appname: []
           user: [1x1 com.bloomberglp.blpapi.impl.aT]
        session: [1x1 com.bloomberglp.blpapi.Session]
      ipaddress: {'111.11.11.112'}
           port: 8194.00
        timeout: 0
```

`bpipe` connects to Bloomberg B-PIPE at the port number `8194`. `bpipe` creates the Bloomberg B-PIPE connection object `c`.

Close the B-PIPE connection.

```
close(c)
```

### Create the Bloomberg Data License Connection

Create the Bloomberg Data License connection. This code assumes the following:

- The Bloomberg Data License SFTP server login name is `'dl338'`.
- The Bloomberg Data License SFTP server password is `'Lb=cYaZ'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'pDyJaV'`.

```
username = 'dl338';
password = 'Lb=cYaZ';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'pDyJaV';

c = bdl(username,password,hostname,portnumber,decrypt)

c =

bdl with properties:

          Login: 'dl338'
       Hostname: 'dlsftp.bloomberg.com'
           Port: 30206
     AuthOption: 'password'
        KeyFile: ''
     Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

`bdl` connects to Bloomberg Data License at port number `30206` with password authentication. `bdl` creates the Bloomberg Data License connection object `c`.

Close the Bloomberg Data License connection.

```
close(c)
```

## See Also

`bdl` | `blp` | `blpsrv` | `bpipe` | `close` | `get` | `isconnection`

## Related Examples

- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Bloomberg Intraday Tick Data” on page 3-12
- “Retrieve Bloomberg Real-Time Data” on page 3-14
- “Retrieve Data Using Bloomberg Data License” on page 3-16

## More About

- “Data Server Connection Requirements” on page 1-3
- “Comparing Bloomberg Connections” on page 2-3
- “Workflow for Bloomberg” on page 3-19

# Retrieve Bloomberg Current Data

This example shows how to retrieve current data from Bloomberg for a single security and for multiple securities. To create a successful Bloomberg connection, see “Connect to Bloomberg” on page 3-2.

### Connect to Bloomberg

Create a Bloomberg Desktop connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

### Retrieve Current Data for Single Security

Retrieve last and open prices for Microsoft®.

`d` contains the Bloomberg last and open prices as fields in a structure. `sec` contains the Bloomberg security name for Microsoft in a cell array. The security name is a character vector.

```
sec = 'MSFT US Equity';
fields = {'LAST_PRICE';'OPEN'}; % Retrieve data for last and open prices

[d,sec] = getdata(c,sec,fields)
```

```
d =

  struct with fields:

    LAST_PRICE: 62.30
          OPEN: 62.95


sec =

  cell

    'MSFT US Equity'
```

### Retrieve Current Data for Multiple Securities

Retrieve last and open prices for the IBM and Ford Motor Company® securities.

`d` contains the Bloomberg last and open prices as fields in a structure. `sec` contains the Bloomberg security names for IBM and Ford Motor Company in a cell array. Each security name is a character vector.

```
s = {'IBM US Equity','F US Equity'};
fields = {'LAST_PRICE';'OPEN'}; % Retrieve data for last and open prices

[d,sec] = getdata(c,s,fields)
```

```
d =

  struct with fields:
```

```
    LAST_PRICE: [2×1 double]
          OPEN: [2×1 double]


sec =

  2×1 cell array

    'IBM US Equity'
    'F US Equity'
```

Display the last price for both securities.

```
d.LAST_PRICE

ans =

        166.73
         12.63
```

**Close Bloomberg Connection**

```
close(c)
```

## See Also

`blp` | `close` | `getdata`

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Bloomberg Intraday Tick Data” on page 3-12
- “Retrieve Bloomberg Real-Time Data” on page 3-14

## More About

- “Workflow for Bloomberg” on page 3-19

# Retrieve Bloomberg Historical Data

This example shows how to retrieve historical data from Bloomberg for a single security. The example shows retrieving weekly data within a date range and retrieving data with a default period. Then, the example also shows how to retrieve data for multiple securities. To create a successful Bloomberg connection, see “Connect to Bloomberg” on page 3-2.

### Connect to Bloomberg

Create a Bloomberg Desktop connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

### Retrieve Historical Data for One Security

Retrieve monthly closing and open price data from January 1, 2012, through December 31, 2012, for Microsoft.

```
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
fromdate = '1/01/2012';
todate = '12/31/2012';
period = 'monthly';

[d,sec] = history(c,s,f,fromdate,todate,period)

d =

     734899.00         27.87         25.06
     734928.00         30.16         28.12
     734959.00         30.65         30.34
     ...

sec =

  cell

    'MSFT US Equity'
```

`d` contains the numeric representation of the date in the first column, closing price in the second column, and open price in the third column. Each row represents data for one month in the date range. `sec` contains the Bloomberg security name for Microsoft.

### Retrieve Weekly Historical Data

Retrieve the weekly closing prices from November 1, 2010 through December 23, 2010 for the Microsoft security using US currency. In this case, the anchor date depends on the date December 23, 2010. Because this date is a Thursday, each previous value is reported for the Thursday of the week in question.

```
f = 'LAST_PRICE';
fromdate = '11/01/2010';
todate =  '12/23/2010';
period = {'weekly'};
currency = 'USD';
```

```
[d,sec] = history(c,s,f,fromdate,todate, ...
    period,currency)

d =

     734446.00         27.14
     734453.00         26.68
     734460.00         25.84
     734467.00         25.37
     734474.00         26.89
     734481.00         27.08
     734488.00         27.99
     734495.00         28.30


sec =

  1×1 cell array

    {'MSFT US Equity'}
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the Microsoft security.

### Retrieve Historical Data Using Default Period

Retrieve the closing prices from August 1, 2010 through September 10, 2010 for the Microsoft security in US currency, and set the default period of the data by using `[]`. The default period of a security depends on the security itself.

```
fromdate = '8/01/2010';
todate = '9/10/2010';
currency = 'USD';

[d,sec] = history(c,s,f,fromdate,todate, ...
    [],currency)

d =

     734352.00         26.33
     734353.00         26.16
     734354.00         25.73
     ...

sec =

  1×1 cell array

    {'MSFT US Equity'}
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the Microsoft security.

### Retrieve Historical Data for Multiple Securities

Retrieve monthly closing and open prices from January 1, 2012, through December 31, 2012, for the IBM and Ford Motor Company securities.

d is a cell array of double matrices that contains the historical data for both securities. sec contains the Bloomberg security names for the IBM and Ford Motor Company securities in a cell array. Each security name is a character vector.

```
s = {'IBM US Equity','F US Equity'};
f = {'LAST_PRICE';'OPEN'};
fromdate = '1/01/2012';
todate = '12/31/2012';
period = 'monthly';

[d,sec] = history(c,s,f,fromdate,todate,period)

d =

  2×1 cell array

    [12×3 double]
    [12×3 double]


sec =

  2×1 cell array

    'IBM US Equity'
    'F US Equity'
```

Display the closing and open prices for the first security.

```
d{1}

ans =

     734899.00        192.60        186.73
     734928.00        196.73        193.21
     734959.00        208.65        197.23
     ...
```

The data in the double matrix is:

- First column — Numeric representation of the date
- Second column — Closing price
- Third column — Open price

Each row represents data for one month in the date range.

#### Close Bloomberg Connection

```
close(c)
```

## See Also

blp | close | history

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6

- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Bloomberg Intraday Tick Data” on page 3-12
- “Retrieve Bloomberg Real-Time Data” on page 3-14

## More About

- “Workflow for Bloomberg” on page 3-19

# Retrieve Bloomberg Intraday Tick Data

This example shows how to retrieve intraday tick data from Bloomberg. To create a successful Bloomberg connection, see “Connect to Bloomberg” on page 3-2.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the trade tick series for the past 50 days for the IBM security aggregated into 5-minute intervals.

```
d = timeseries(c,'IBM US Equity',{floor(now)-50,floor(now)},5,'Trade')
```

```
ans =

  Columns 1 through 7

   735487.40        187.20        187.60        187.02        187.08     207683.00        560.00
   735487.40        187.03        187.13        186.65        186.78      46990.00        349.00
   735487.40        186.78        186.78        186.40        186.47      51589.00        399.00
     ...

Column 8

   38902968.00
    8779374.00
    9626896.00
   ...
```

The columns in `d` contain the following:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row of data shows prices and tick data for the current date. The next row shows tick data for 5 minutes later.

Close the Bloomberg connection.

```
close(c)
```

## See Also

`blp` | `close` | `timeseries`

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Bloomberg Real-Time Data” on page 3-14

## More About

- “Workflow for Bloomberg” on page 3-19

# Retrieve Bloomberg Real-Time Data

This example shows how to retrieve real-time data from Bloomberg. To create a successful Bloomberg connection, see “Connect to Bloomberg” on page 3-2. Here, to return Bloomberg stock tick data, use the event handler `v3stockticker`. Instead of the default event handler, you can create your own event handler function to process Bloomberg data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume for IBM and Ford Motor Company securities.

`v3stockticker` requires the input argument `f` of the `realtime` function to be `'Last_Trade'`, `'Volume'`, or both.

```
[subs,t] = realtime(c,{'IBM US Equity','F US Equity'},...
                    {'Last_Trade','Volume'},'v3stockticker')
```

```
subs =

com.bloomberglp.blpapi.SubscriptionList@6c1066f6


   Timer Object: timer-3

   Timer Settings
      ExecutionMode: fixedRate
             Period: 0.05
           BusyMode: drop
            Running: on

   Callbacks
           TimerFcn: 1x4 cell array
           ErrorFcn: ''
           StartFcn: ''
            StopFcn: ''

** IBM US Equity ** 32433 @ 181.85 29-Oct-2013 15:50:05
** IBM US Equity ** 200 @ 181.85 29-Oct-2013 15:50:05
** IBM US Equity ** 100 @ 181.86 29-Oct-2013 15:50:05
** F US Equity ** 300 @ 17.575 30-Oct-2013 10:14:06
** F US Equity ** 100 @ 17.57 30-Oct-2013 10:14:06
** F US Equity ** 100 @ 17.5725 30-Oct-2013 10:14:06
...
```

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `realtime` returns the stock tick data for the IBM and Ford Motor Company securities with the last trade price and volume.

Real-time data continues to display until you use the `stop` or `close` function.

Close the Bloomberg connection.

```
close(c)
```

## See Also

`blp` | `close` | `realtime` | `stop`

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-14
- “Retrieve Bloomberg Intraday Tick Data” on page 3-12

## More About

- “Workflow for Bloomberg” on page 3-19
- “Writing and Running Custom Event Handler Functions” on page 1-26

# Retrieve Data Using Bloomberg Data License

This example shows how to retrieve Bloomberg Data License data with a request file using a Bloomberg Data License connection. To create a successful Bloomberg connection, see “Connect to Bloomberg” on page 3-2. To access the code for this example, enter `edit BloombergDataLicenseWorkflow.m`.

### Connect to Bloomberg Data License

Create the Bloomberg Data License connection `c`. This code assumes the following:

- The Bloomberg Data License SFTP server login name is `'xxxxx'`.
- The Bloomberg Data License SFTP server password is `'xxxxxxxx'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.

```
username = 'xxxxx';
password = 'xxxxxxxx';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'nAcLeZ';

c = bdl(username,password,hostname,portnumber,decrypt);
```

`bdl` connects to Bloomberg Data License at port number `30206` with password authentication.

### Request Bloomberg Data File

Create a Bloomberg Data License request file `getdatarequest.req` using the Bloomberg Data License Request Builder. Submit the request file to Bloomberg Data License using `c`.

```
c.Connection.put('getdatarequest.req')
```

Retrieve the folder listing to see if the output file exists using `c`.

```
s = dir(c)
```

```
s =

     'd--x--x--x    2 root     root         4096 Sep  5 11:25 bin'
     'dr--r--r--    2 root     root         4096 Sep  5 11:25 etc'
     '-rw-rw-rw-    2 op       general      1194 Sep 24 10:14 getdataoutput.out'
     ...
```

The output file `getdataoutput.out` is available.

Save the output file to the current folder. The first argument is the name of the generated output file from the Bloomberg Data License server. The second argument is the name of the saved file on the local machine.

```
c.Connection.get('getdataoutput.out','getdataoutput.out')
```

The current folder contains the output file `getdataoutput.out`.

If the file is encrypted, run the Bloomberg Data License decryption software. You can download the decryption software from Bloomberg. For questions, contact Bloomberg.

```
!des -D -u -k "pDyJaV" getdataoutput.out  getdataoutput.dat
```

**Import Bloomberg Data into MATLAB**

Convert the contents of the output file to a MATLAB structure using the sample function `bdlloader`.

```
d = bdlloader('getdataoutput.out')
```

```
d =

          Header: [1x1 struct]
      Identifier: {4x1 cell}
           Rcode: {4x1 cell}
         Nfields: {4x1 cell}
         PX_OPEN: {4x1 cell}
         PX_LAST: {4x1 cell}
         PX_HIGH: {4x1 cell}
          PX_LOW: {4x1 cell}
     PX_CLOSE_DT: {4x1 cell}
```

`d` is a structure with these fields:

- Output file header information
- Security identifier
- Return code
- Number of fields requested and received
- Open price
- Last price
- High price
- Low price
- Date of last close

To access the code for `bdlloader`, see `bdlloader.m`.

Display the output file header information.

```
d.Header
```

```
ans =

            RUNDATE: '20140924'
        PROGRAMFLAG: 'oneshot'
           FIRMNAME: 'xxxxx'
           FILETYPE: 'pc'
      REPLYFILENAME: 'getdataoutput.out'
     PRICING_SOURCE: 'BVAL'
      CLOSINGVALUES: 'yes'
              SECID: 'TICKER'
         YELLOWKEY: 'Equity'
        PROGRAMNAME: 'getdata'
        TIMESTARTED: 'Wed Sep 24 10:19:59 EDT 2014'
       TIMEFINISHED: 'Wed Sep 24 10:20:17 EDT 2014'
```

### Close Bloomberg Data License Connection

```
close(c)
```

## See Also

`bdl` | `close` | `dir`

## Related Examples

- “Connect to Bloomberg” on page 3-2

## More About

- “Workflow for Bloomberg” on page 3-19

# Workflow for Bloomberg

| In this section... |
| --- |
| “Bloomberg Desktop, Bloomberg Server, or Bloomberg B-PIPE Services” on page 3-19 |
| “Bloomberg Data License Service” on page 3-19 |

You can use Bloomberg to monitor market price information.

## Bloomberg Desktop, Bloomberg Server, or Bloomberg B-PIPE Services

### Connect to Bloomberg

1. Connect to Bloomberg using `blp`, `blpsrv`, or `bpipe`.
2. Ensure a successful Bloomberg connection by using `isconnection`. Request properties of the connection objects using `get`.

### Request Current, Historical, Intraday Tick, Portfolio, or Real-Time Data

1. Look up information about securities, curves, or government securities using `lookup`. Request Bloomberg field information using `category`, `fieldinfo`, or `fieldsearch`.
2. Request current data for a security using `getdata`. Request bulk data with header information using `getbulkdata`.
3. Request equity screening data using `eqs`.
4. Request historical data for a security using `history`.
5. Request historical technical analysis using `tahistory`.
6. Request intraday tick data for a security using `timeseries`.
7. Request current portfolio data using `portfolio`.
8. Request real-time data for a security using `realtime`. Stop real-time data updates using `stop`.

### Close Bloomberg Connection

Close the Bloomberg connection by using `close`.

## Bloomberg Data License Service

To connect and retrieve data using Bloomberg Data License:

1. Connect to Bloomberg Data License using `bdl`.
2. Request the current Bloomberg Data License folder listing using `dir`.
3. Close the Bloomberg connection by using `close`.

## See Also

## Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6

- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Bloomberg Intraday Tick Data” on page 3-12
- “Retrieve Bloomberg Real-Time Data” on page 3-14
- “Retrieve Data Using Bloomberg Data License” on page 3-16

## More About

- “Comparing Bloomberg Connections” on page 2-3

# Determine the Event Volume Indicator Using RavenPack News Analytics

This example shows how to determine the Event Volume Indicator (EVI) from RavenPack News Analytics historical and intraday data. The example also shows how to retrieve real-time data to update the EVI.

The EVI counts the number of RavenPack News Analytics news events based on the event sentiment score (ESS). Find the EVI on day $t$ using

$$\mathrm{EVI}_t = \sum_{i=0}^{n-1} (pos_{t-i} + neg_{t-i}) \ .$$

The variables are:

- $pos_t$ is the count of positive events on day $t$.
- $neg_t$ is the count of negative events on day $t$.
- $n$ is the number of days for which you are calculating the EVI.

ESS is the Event Sentiment Score that measures positive or negative news sentiment. Define positive events by ESS > 50 and negative events by ESS < 50. For details about ESS, see the *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview. For details about this calculation, see [1].

To analyze the news activity about a company or other entity, use the EVI.

To access the code for this example, see `RavenPackWorkflowExample.m`.

### Connect to RavenPack News Analytics

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

### Retrieve Historical and Intraday RavenPack News Data

Retrieve RavenPack News Analytics Data Gateway entitlements using the RavenPack News Analytics connection `c`.

```
e = entitlements(c);
```

`e` is a table that contains the entitlement data.

Retrieve the RavenPack News Analytics symbol for equities data.

```
symbol = e.NAME{1};
```

Create a historical equities data file `2014-11-equities.csv` using the RavenPack Data Feed Tool. Load the data into the MATLAB variable `histData`.

```
histData = rploader('2014-11-equities.csv');
```

Request the last two and a half hours of intraday data `recentData` for all valid fields using the RavenPack News Analytics connection `c`. `symbol` is the entitled symbol for equity data. Create the time interval using `now-.1` and `now`.

```
recentData = timeseries(c,symbol,{now-.1,now});
```

To retrieve the last minute of intraday data for the RavenPack News Analytics fields `ENTITY_NAME`, `GROUP`, and `ESS`, use this code. Create the time interval for the last minute using `now-.001` and `now`.

```
essData = timeseries(c,symbol,{now-.001,now},...
{'ENTITY_NAME','GROUP','ESS'});
```

Combine the historical and intraday data into the MATLAB variable `allData`. Remove the time component from the timestamp using `dateshift`.

```
allData = [histData; recentData];
allData.TIMESTAMP_UTC = dateshift(allData.TIMESTAMP_UTC,'start','day');
```

### Retrieve Twitter News Data

Isolate the Twitter® news event data using `strcmp`. Assign the Twitter news event data to the MATLAB variable `companyData`.

```
iCompany = strcmp('Twitter Inc.',allData.ENTITY_NAME);
companyData = allData(iCompany,:);
```

Filter out the insider trading and order imbalance events from the Twitter news event data. To find these events, use the RavenPack News Analytics field `GROUP`. For details about this field, see RavenPack Developer Zone Overview. For details about this filter, see [1].

```
iIT = strcmp('insider-trading',companyData.GROUP);
companyData(iIT,:) = [];
iIT = strcmp('order-imbalances',companyData.GROUP);
companyData(iIT,:) = [];
```

### Determine EVI for Twitter News Data

This code loops through the Twitter company data `companyData`. The logic filters out empty ESS scores and ESS scores equal to 50. The code adds a record for the first event of the day using the timestamp `essTimestamp`. Then, the code increments the EVI count `eviData`.

```
EVI = 0;
eviData(1,1) = EVI;
essTimestamp = companyData.TIMESTAMP_UTC(1);
iEVI = 1;
for j = 1:length(companyData.ESS)
  % Discard neutral and empty scores
  if ~isempty(companyData.ESS{j}) && ~strcmp(companyData.ESS{j},'50')
    % Add new record for new day's event, otherwise increment existing count
    if essTimestamp(iEVI) ~= companyData.TIMESTAMP_UTC(j)
      essTimestamp(end+1,1) = companyData.TIMESTAMP_UTC(j);
      iEVI = length(essTimestamp);
      EVI = 0;
    else
      essTimestamp = companyData.TIMESTAMP_UTC(j);
    end
    EVI = EVI + 1;
    eviData(iEVI,1) = EVI;
  end
end
```

Populate a table `twitter` with a timestamp `essTimestamp` and the Twitter EVI data `eviData`.

```
twitter = table(essTimestamp,eviData,...
'VariableNames',{'Timestamp','Twitter_EVI'});
```

Clear temporary MATLAB variables.

```
clear essTimestamp EVI eviData
```

#### Retrieve Real-Time Twitter News Data

Populate real-time Twitter EVI data into the MATLAB variable `twitter` in the Workspace browser using `realtime`. Run `realtime` using the RavenPack News Analytics connection `c` and symbol. The sample listener `rpExampleListener` listens for news event data from any company or entity. When `realtime` is run as part of this example, `realtime` provides real-time updates for the Twitter EVI. Here, this listener is monitoring for news related to these RavenPack News Analytics fields: `ENTITY_NAME`, `GROUP`, and `ESS`. To add other functionality, you can modify this listener function or create your own.

```
[status,lhandle] = realtime(c,symbol, ...
    @(~,evt)rpExampleListener(evt,{'ENTITY_NAME', ...
    'GROUP','ESS'}));
```

#### Close RavenPack News Analytics Connection

```
close(c)
```

## References

[1] Hafez, Peter, and Junqiang Xie. “Enhancing Short Term Reversal Strategies with News Analytics.” *RavenPack Quantitative Research The News Analytics Specialist*. September 5, 2013, p. 3.

## See Also

`close` | `entitlements` | `ravenpack` | `realtime` | `rploader` | `timeseries`

## More About

- “Workflow for RavenPack News Analytics” on page 3-24

## External Websites

- RavenPack Developer Zone Overview

# Workflow for RavenPack News Analytics

You can use RavenPack News Analytics to retrieve news data.

To request intraday, historical, or real-time news data:

1. Connect to RavenPack News Analytics using `ravenpack`.
2. Retrieve RavenPack News Analytics entitlements using `entitlements`.
3. Retrieve intraday and historical news data using `timeseries`.
4. Retrieve real-time news data using `realtime`.
5. Read RavenPack News Analytics data in a file using `rploader`.
6. Close the RavenPack News Analytics connection using `close`.

## See Also

## Related Examples

- “Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21

# Compare Player Salaries by Injury Status

This example shows how to retrieve salary and injury data from STATS.com for individual baseball players. To compare salaries of players with and without injuries, visualize the data in a scatter plot.

This example requires a STATS.com API key and secret pass code for baseball data. For details about your credentials for data access, contact STATS.com.

To access the code for this example, enter `edit MLBSalaryVsInjuryExample.m`.

### Connect to STATS.com

Create a STATS.com connection `sBaseball` for retrieval of statistical baseball data for the sport league named `'mlb'`. Set up the connection to retrieve salary data by specifying the query parameter `'Resource'` as `'salaries'`.

```
sBaseball = statsllc('gkfrq6fxabcehmn2yctrc6j5','qYR5abCQgc', ...
    'DataType','stats','LeagueAbbreviation','mlb', ...
    'Resource','salaries','SportName','baseball', ...
    'VersionNumber','v1');
```

### Retrieve Baseball Player Salary and Injury Data

Retrieve individual baseball player salaries `data` using `sBaseball`.

```
data = fetch(sBaseball);
```

To retrieve the individual baseball player data `playerData`, access the nested structure `data`. Create a salary data table `salaryData` by retrieving the structure fields in `playerData`.

```
playerData = [data.apiResults.league.season.salaries.player];

salaryData = table((1:data.recordCount)',[playerData.playerId]', ...
    {playerData.firstName}',{playerData.lastName}', ...
    [data.apiResults.league.season.salaries.salary]', ...
    false(data.recordCount,1), ...
    'VariableNames',{'Record','PlayerId','FirstName', ...
    'LastName','Salary','Injured'});
```

`salaryData` contains these column names:

- `Record` — Record number
- `PlayerId` — Baseball player identifier
- `FirstName` — Baseball player first name
- `LastName` — Baseball player last name
- `Salary` — Baseball player salary
- `Injured` — Baseball player injured status

Baseball player injured status is a placeholder for the injury data.

To retrieve the injury data, change the query parameter `Resource` value from `'salaries'` to `'participants'` in the STATS.com connection `sBaseball`. Retrieve the individual baseball player data using `sBaseball`.

```
pause(2)
sBaseball.Resource = 'participants';

data = fetch(sBaseball);
```

Loop through the resulting nested structure `data` for each player. Set the injured status in `salaryData` for those players that are injured.

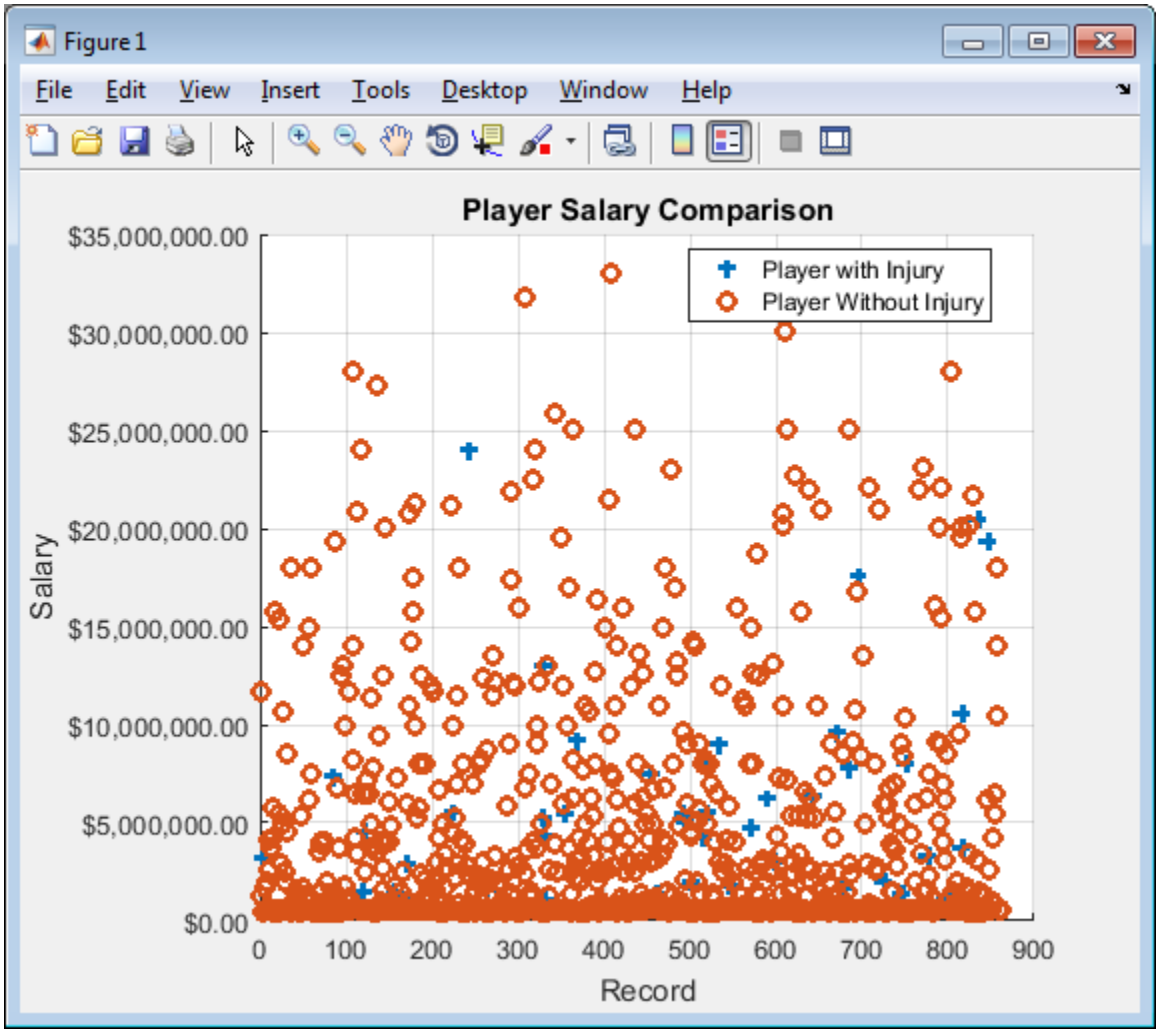
```
numPlayers = data.recordCount;
for i = 1:numPlayers
    playerId = data.apiResults.league.players{i}.playerId;
    salaryData.Injured(salaryData.PlayerId == playerId) ...
        = data.apiResults.league.players{i}.isInjured;
end
```

### Visualize Baseball Player Salary and Injury Data

Create a scatter plot of the salary for each baseball player.

```
sInjured = scatter(salaryData.Record(salaryData.Injured), ...
    salaryData.Salary(salaryData.Injured),'Marker','+', ...
    'Linewidth',2);
hold on
sNotInjured = scatter(salaryData.Record(~salaryData.Injured), ...
    salaryData.Salary(~salaryData.Injured),'Marker','O', ...
    'Linewidth',2);
hold off
sInjured.MarkerFaceColor = 'flat'
grid on
b = gca
title('Player Salary Comparison')
xlabel('Record')
ylabel('Salary')
ytickformat(b,'usd')
legend({'Player with Injury','Player Without Injury'},'Location','best')
```

The scatter plot of baseball players and their salaries displays. Compare the salaries for injured and uninjured baseball players.




## See Also

`fetch` | `statsllc`

## More About

- "Retrieve Team Standings for the Current Year" on page 3-28
- "STATS.com Data Retrieval Errors" on page 4-2

## External Websites

- STATS.com
- STATS Developer Center I/O Docs

# Retrieve Team Standings for the Current Year

This example shows how to retrieve basketball team standing data from STATS.com for the current year.

This example requires a STATS.com API key and secret pass code for basketball data. For details about your credentials for data access, contact STATS.com.

To access the code for this example, enter `edit NBAStandingsExample.m`.

### Connect to STATS.com

Create a STATS.com connection `sBasketball` for retrieval of statistical basketball data for the sport league named `'nba'`. Set up the connection to retrieve team standing data by specifying the query parameter `'Resource'` as `'standings'`.

```
sBasketball = statsllc('se33pc658r9abcahm4a9j93p','frvaBcb33y', ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1');
```

### Retrieve Team Standings Data

Retrieve basketball team standings `data` using `sBasketball`.

```
data = fetch(sBasketball);
```

Retrieve the conference, division, and team data by accessing the nested structure `data`. To access each layer of the data, create a nested loop. Convert the returned structure data to tables. Create the table `standingsTable` that contains the team standings data. Display the data in the Command Window.

```
numConferences = length(data.apiResults.league.season.eventType.conferences);

% Iterate through each conference
for i = 1:numConferences
    numDivisions = length(data.apiResults.league.season.eventType.conferences(i).divisions);

    % Iterate through each division
    for j = 1:numDivisions

        % Access team data in each division
        t = struct2table(data.apiResults.league.season.eventType.conferences(i).divisions(j).teams);
        numTeams = length(data.apiResults.league.season.eventType.conferences(i).divisions(j).teams);
        recordTable = [];

        % Iterate through each team
        for k = 1:numTeams
            divisionStatusTable = struct2table(data.apiResults.league.season.eventType.conferences(i).divisions(j).teams(k).division);
            recordTable = [recordTable;struct2table(data.apiResults.league.season.eventType.conferences(i).divisions(j).teams(k).record
                table(divisionStatusTable.gamesBehind, ...
                    'VariableNames',{'GamesBack'})];
        end

        % Create table of team standing data
        standingsTable = table(strcat(t.location,strcat('.',t.nickname)), ...
            recordTable.wins,recordTable.losses,recordTable.percentage, ...
            recordTable.GamesBack,'VariableNames', ...
            {'Team','Wins','Losses','Percentage','GamesBehind'});

        % Display team standing data to the command line
        disp([data.apiResults.league.season.eventType.conferences(i).name ...
            ' ' data.apiResults.league.season.eventType.conferences(i).divisions(j).name])
        disp(' ')
        disp(standingsTable)
    end
end
```

```
Eastern Conference Atlantic Division

            Team             Wins    Losses    Percentage    GamesBehind
    _____________________    ____    ______    __________    ___________

    'Toronto.Raptors'        56      26        .683           0
    'Boston.Celtics'         48      34        .585           8
    'New York.Knicks'        32      50        .390          24
    'Brooklyn.Nets'          21      61        .256          35
    'Philadelphia.76ers'     10      72        .122          46
...
```

The Command Window displays the basketball team standings for each division. The column names are:

- `Team` — Basketball team name
- `Wins` — Number of wins
- `Losses` — Number of losses
- `Percentage` — Percentage of games won
- `GamesBehind` — Number of games behind

## See Also

`fetch` | `statsllc`

## More About

- “Compare Player Salaries by Injury Status” on page 3-25
- “STATS.com Data Retrieval Errors” on page 4-2

## External Websites

- STATS.com
- STATS Developer Center I/O Docs

# Troubleshooting

# STATS.com Data Retrieval Errors

To return STATS.com data, the STATS.com web service uses a URL for making a web request. This table describes how to address common errors returned from the web service when you encounter issues with the web request.

| STATS.com Web Service Error | Probable Causes | Resolution |
|---|---|---|
| The server returned the message: "Internal Server Error" for URL, *`url`* (with HTTP response code 500). | The server connection is not established. | Reconnect to STATS.com using `statsllc`. |
| The server returned the message: "Forbidden" for URL, *`url`* (with HTTP response code 403). | You are retrieving data for query parameters that are not included in your license. Or, you have exceeded the number of allowable requests that are included in your license. | • Verify your credentials: the API key and secret pass code. Then, reconnect to STATS.com using `statsllc`.<br>• Check your STATS.com license for the query parameters that are included and the number of allowable requests in a specific time frame. |
| The server returned the message: "Uknown" for URL, *`url`* (with HTTP response code 596). | • The URL suffix input argument in `fetchUrl` is invalid.<br>• The `statsllc` object has invalid or missing property values.<br>• The query parameters input argument in `fetch` is invalid. | • Verify the URL suffix input argument in `fetchUrl`.<br>• Verify the object properties in the `statsllc` object. If there are invalid properties, create a STATS.com connection using `statsllc`. If there are missing property values, set the object property values. For details about setting object properties, see `statsllc`.<br>• Verify the query parameters. Check which query parameters are included in your STATS.com license. |

## See Also

## Related Examples

- “Compare Player Salaries by Injury Status” on page 3-25
- “Retrieve Team Standings for the Current Year” on page 3-28

## External Websites

- STATS.com

- STATS Developer Center I/O Docs

# Money.Net Topics


- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-5
- “Retrieve Money.Net News Stories” on page 5-7
- “Money.Net Error and Warning Messages” on page 5-10

# Retrieve Current and Historical Money.Net Data

This example shows how to retrieve current data for symbols, historical data, and current data for option symbols from Money.Net.

To run this example, you need a Money.Net user name and password. To request these credentials, contact Money.Net.

To access the code for this example, enter `edit MoneyNetDataWorkflowExample.m`.

### Create Money.Net Connection

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

### Retrieve Money.Net Current Data

Retrieve Money.Net current data `d` for the symbol IBM using the Money.Net connection `c`. Specify the Money.Net data fields `f` for ask and bid price.

```
symbol = 'IBM';
f = {'Ask','Bid'};

d = getdata(c,symbol,f);
```

Display Money.Net current data. `d` is a table that contains the variables for symbol, ask price, and bid price. The row contains Money.Net data values for each variable.

```
d
```

```
d =

    Symbol     Ask       Bid
    ______    ______    ______

    'IBM'     145.00    143.85
```

Retrieve Money.Net current data for the `symbols` list that contains these symbols: IBM, Google, and Yahoo!®.

```
symbols = {'IBM','GOOG','YHOO'};

d = getdata(c,symbols,f);
```

Display Money.Net current data. `d` is a table that contains the variables for symbol, ask price, and bid price. The rows contain Money.Net data values for each symbol in the symbol list.

```
d
```

```
d =

    Symbol     Ask       Bid
    ______    ______    ______
```

```
    'IBM'     145.00    143.85
    'GOOG'    700.50    700.05
    'YHOO'     37.50     37.41
```

### Retrieve Money.Net Historical Data

Retrieve historical data in daily bars for the symbol IBM. Specify the date range from June 1, 2015 through June 5, 2015 using `datetime`. To retrieve daily data, specify the interval as `'1D'`. Retrieve only the high and low price fields `f` from Money.Net.

`d` is a table that contains these variables:

- Date timestamp
- High price
- Low price

```
s = 'IBM';
date = [datetime('1-Jun-2015') datetime('5-Jun-2015')];
interval = '1D';
f = {'High','Low'};

d = timeseries(c,s,date,interval,f);
```

Display the first three rows of daily data `d`.

```
d(1:3,:)
```

```
ans =

          Date             High      Low
    ___________________    ______    ______

    06/01/15 00:00:00    171.04    169.03
    06/02/15 00:00:00    170.45    168.43
    06/03/15 00:00:00    171.56    169.63
```

Determine the average high price in the date range.

```
mean(d.High)
```

```
ans =

        170.51
```

### Retrieve Money.Net Option Symbol Data

Retrieve option symbols `o` for the symbol IBM. `o` is a cell array of character vectors. Each character vector is an option symbol.

```
s = 'IBM';

o = optionchain(c,s);
```

Display the first three option symbols.

```
o(1:3)
```

```
ans =
```

```
3×1 cell array

  'O:IBM\16F24\131 .0'
  'O:IBM\16R24\135 .0'
  'O:IBM\16F24\142 .0'
```

Retrieve the current data for the first option symbol `o(1)` and display it. Specify fields `f` for describing the option symbol:

- Option symbol description
- Option symbol strike
- Option symbol expiration date
- Option symbol ask price
- Option symbol bid price

`d` is a table with one row of data. The data contains the option symbol name in the first variable and a variable for each specified field `f`.

```
symbol = o(1);
f = {'Description','Strike','Expiration','Ask','Bid'};

d = getdata(c,symbol,f)
```

```
d =

          Symbol                Description             Strike    Expiration     Ask      Bid
    ____________________    ____________________________    ______    __________    _____    _____

    'O:IBM\16F24\131 .0'    'IBM Call 06/24/2016 131.0'    131       06/24/16      23.75    21.75
```

### Close Money.Net Connection

```
close(c)
```

## See Also

`close` | `getdata` | `moneynet` | `optionchain` | `timeseries`

## More About

- “Retrieve Real-Time Money.Net Data” on page 5-5
- “Retrieve Money.Net News Stories” on page 5-7
- “Money.Net Error and Warning Messages” on page 5-10

## External Websites

- Money.Net API Documentation
- Money.Net Help

# Retrieve Real-Time Money.Net Data

This example shows how to retrieve real-time data from Money.Net for a symbol. It explains how to subscribe to real-time updates, stop subscription, and process real-time updates using a custom event handler function.

To process real-time data updates, you can use the default event handler function. Or, for a different approach, you can write a custom event handler function. For writing custom event handler functions with Money.Net data, see `realtime`. For custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-26.

This example requires a Money.Net user name and password. To request these credentials, contact Money.Net.

To access the code for this example, enter `edit MoneyNetDataWorkflowExample.m`.

### Create Money.Net Connection

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

### Retrieve Real-Time Data for One Symbol

Retrieve Money.Net real-time data updates for the IBM symbol.

```
symbol = 'IBM';

realtime(c,symbol)
```

The default event handler `mnRealTimeEventHandler` processes all real-time data updates. To access the code for the default event handler, enter `edit mnRealTimeEventHandler.m`.

The `mnRealTimeEventHandler` function creates the workspace variable `IBMRealTime`. The `mnRealTimeEventHandler` function populates the table `IBMRealTime` with real-time data updates. To see the real-time data, open `IBMRealTime` in the Variables editor.

### Stop Real-Time Data Updates

Stop the symbol subscription.

```
stop(c)
```

`mnRealTimeEventHandler` stops processing all real-time data updates. The last real-time data update remains in `IBMRealTime`.

### Retrieve Real-Time Data Using Custom Event Handler Function

Define a custom event handler function `myfcn`. The `myfcn` function displays Money.Net real-time data to the Command Window.

```
myfcn = @(x)disp(x);
```

Retrieve Money.Net real-time data updates for the IBM symbol using `myfcn`.

```
symbol = 'IBM';

realtime(c,symbol,myfcn)
```

| Symbol | Description | Yesterday | YesterdayDateTime | Bid | Ask | ExchangeOfTheCurrentBidPrice | Ex |
|---|---|---|---|---|---|---|---|
| 'IBM' | 'INTERNATIONAL BUSINESS MACHS' | 148.31 | 05/24/16 00:00:00 | 151.65 | 151.67 | '' | '' |

`myfcn` displays real-time data updates for IBM in the Command Window.

Stop the symbol subscription.

```
stop(c,symbol)
```

`myfcn` stops displaying real-time data updates in the Command Window.

#### Close Money.Net Connection

```
close(c)
```

## See Also

`close` | `moneynet` | `realtime` | `stop`

## More About

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Money.Net News Stories” on page 5-7
- “Writing and Running Custom Event Handler Functions” on page 1-26
- “Money.Net Error and Warning Messages” on page 5-10

## External Websites

- Money.Net API Documentation
- Money.Net Help

# Retrieve Money.Net News Stories

This example shows how to retrieve news stories from Money.Net in different ways. You can search for a specific number of news stories. You can search for news stories using specific filter criteria. Or, you can stream news stories in real time.

To run this example, you need a Money.Net user name and password. To request these credentials, contact Money.Net.

To access the code for this example, enter `edit MoneyNetNewsWorkflowExample.m`.

### Create Money.Net Connection

Create the Money.Net connection `c` using the user name `username` and password `pwd`.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

### Retrieve Specific Number of News Stories

Retrieve news data `n` for 10 news stories using the Money.Net connection `c`.

```
n = news(c,'Number',10);
```

Display the news story title, identifier, and published time for the first news story in the table `n`.

```
n(1,1:3)
```

```
ans =
```

| ArticleTitle | ArticleID | PublishedTime |
|---|---|---|
| 'Stop talking about replacements. Give PC owners something new al…' | 3.8917e+09 | 05/13/16 10:00:02 |

### Search News Stories Using Search Term

Retrieve news stories that mention the term Windows. `n` is a table with data for 50 news stories.

```
term = 'Windows';

n = news(c,'SearchTerm',term);
```

Display the news story title, identifier, and published time for the first news story.

```
n(1,1:3)
```

```
ans =
```

| ArticleTitle | ArticleID | PublishedTime |
|---|---|---|
| 'LogMein Shares Edge Lower; LastPass Says Browser Extension Now A…' | 4.0005e+09 | 06/08/16 13:22:04 |

### Search News Stories Using Category

Retrieve news stories in the general finance category. `n` is a table with data for 50 news stories.

```
category = 'General Finance';

n = news(c,'Category',category);
```

Display the news story title, identifier, and published time for the first news story.

```
n(1,1:3)
```

```
ans =
                              ArticleTitle                                ArticleID       PublishedTime
    ______________________________________________________________________    _________    ________________

    'Keep calm and ooze compassion: Leave must seize the moral high g…'    4.0007e+09    06/08/16 12:48:42
```

### Search News Stories Using Symbol

Retrieve news stories that contain the symbol for Microsoft. `n` is a table with data for 50 news stories.

```
symbol = 'MSFT';

n = news(c,'Symbol',symbol);
```

Display the news story title, identifier, and published time for the first news story.

```
n(1,1:3)
```

```
ans =
                    ArticleTitle                       ArticleID       PublishedTime
    ___________________________________________________    _________    ________________

    'Microsoft announces after party to Apple's WWDC'    4.0005e+09    06/08/16 12:51:49
```

### Analyze News Stories for Analyst Ratings

Search the analyst ratings category for Microsoft. Return 100 news stories.

```
symbol = 'MSFT';
category = 'Analyst Ratings';

n = news(c,'Number',100,'Symbol',symbol,'Category',category);
```

Convert news story titles to the string array `titles`.

```
titles = string(n.ArticleTitle);
```

Perform a non-case-sensitive search of the titles using `contains`. Here, assume that the word `'buy'` represents a buy rating for Microsoft from an investment analyst. Count the occurrences of buy ratings in the 100 news stories.

```
sentiment = contains(titles,'buy','IgnoreCase',true);

sum(sentiment)

ans =

    33
```

To compare buy ratings against sell and hold ratings, replace `'buy'` with the corresponding term and count the occurrences. With these counts, you can see which ratings are more common.

### Stream News Stories in Real Time

Start the subscription to the Money.Net real-time news data stream using the default event handler function `mnNewsStreamEventHandler`. The function `mnNewsStreamEventHandler` processes news data events by populating the workspace variable `mnNewsStreamLatest` with the latest news stories. News stories populate in the `mnNewsStreamLatest` variable until it contains 10 rows. Then, the latest news stories overwrite the older ones in `mnNewsStreamLatest`. To access the code for this function, enter `edit mnNewsStreamEventHandler.m`.

```
news(c,'Subscription','on')
```

The workspace variable `mnNewsStreamLatest` appears in the MATLAB Workspace.

Display the news story title, identifier, and published time for the first news story.

```
mnNewsStreamLatest(1,1:3)
```

```
ans =
                            ArticleTitle                                ArticleID       PublishedTime
    ____________________________________________________________________    _________    ________________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09    05/13/16 10:00:02
```

To see the latest 10 news stories, open `mnNewsStreamLatest` in the Variables editor.

Stop the real-time news data stream.

```
news(c,'Subscription','off')
```

Money.Net stops updating news stories in `mnNewsStreamLatest`.

### Close Money.Net Connection

```
close(c)
```

## See Also

`close` | `contains` | `moneynet` | `news`

## More About

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-5
- “Money.Net Error and Warning Messages” on page 5-10

## External Websites

- Money.Net API Documentation
- Money.Net Help

# Money.Net Error and Warning Messages

| In this section... |
| --- |
| “Money.Net Connection Error Messages” on page 5-10 |
| “Money.Net Data Retrieval Error Messages” on page 5-11 |
| “Money.Net Data Retrieval Warning Messages” on page 5-11 |

Address any error or warning messages that you encounter while connecting to or retrieving data from Money.Net using these tables.

## Money.Net Connection Error Messages

| Connection Error Message | Probable Causes | Resolution |
| --- | --- | --- |
| Failed to connect to the Money.Net data servers. Confirm that the port number is valid. | The specified port number is invalid.<br><br>A firewall or proxy is denying the connection. | To specify the default port number, use the first syntax in `moneynet`. To set up a firewall or proxy to work with Money.Net, see Money.Net Firewall Instructions.<br><br>Your firewall must support a direct internet connection to Money.Net. For details, see Money.Net Firewall Instructions. |
| Invalid username or password. | The user name and password combination is invalid. | Verify your user name and password. To validate these credentials, contact Money.Net. |
| User *`username`* is already logged into Money.Net. | A Money.Net connection is open with the specified user name. Only one connection can exist for each user name at a time. | First, to close the previous Money.Net connection, run the `close` function. Then, to create a Money.Net connection, run the `moneynet` function. |
| Lost connection to the Money.Net data server. Use the MONEYNET command to connect again. | The Money.Net data server interrupts the connection.<br><br>You create multiple `moneynet` objects. | To create a Money.Net connection, run the `moneynet` function. |
| MONEYNET object is not connected to the Money.Net data servers. Use MONEYNET to create a new connection. | You execute a function using a closed Money.Net connection. | To create a Money.Net connection, run the `moneynet` function. |

## Money.Net Data Retrieval Error Messages

| Data Retrieval Error Message | Probable Causes | Resolution |
|---|---|---|
| Unrecognized data field *`Field`*. | The Money.Net field for the field input argument in the `getdata` or `timeseries` functions is invalid. | Verify the Money.Net field name. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation. |
| Unrecognized historical field *`Field`*. | The Money.Net field for the field input argument in the `getdata` or `timeseries` functions is invalid. | Verify the Money.Net field name. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation. |
| Invalid combination of Name-Value pairs. Type HELP MONEYNET/NEWS to see the valid syntax | The combination of name-value pair arguments in the `news` function is invalid. | Verify the name-value pair argument syntax when running `news`. |

## Money.Net Data Retrieval Warning Messages

| Data Retrieval Warning Message | Probable Causes | Resolution |
|---|---|---|
| No timeseries data returned. | No intraday or historical data that matches the specified input arguments in the `timeseries` function is available.<br><br>The specified historical date range is outside of the available dates from Money.Net. Second and minute interval data is only available for more recent dates. | Verify that the symbol and interval input arguments are valid. For details, see `timeseries`.<br><br>Verify that the specified dates contain activity. For example, no data is available on weekends or holidays.<br><br>Specify a recent date range. |
| No option month data returned for *`Symbol`*. | The specified symbol is invalid.<br><br>No options are available for the specified symbol. | Verify the symbol is valid. |
| No news stories found that match the search criteria. | No news stories are available that match the specified criteria in `news`. | Change the search criteria for news stories. For details, see `news`. |

| Data Retrieval Warning Message | Probable Causes | Resolution |
|---|---|---|
| Error occurred while processing real-time data: | The custom event handler function returns an error when processing real-time data. | To find the cause of the error, troubleshoot the custom event handler function code. For writing a custom event handler function using Money.Net data, see `realtime`. For custom event handlers, see “Writing and Running Custom Event Handler Functions” on page 1-26. |
| Error occurred while processing real-time news: | The custom event handler function returns an error when processing real-time news data. | To find the cause of the error, troubleshoot the custom event handler function code. For writing a custom event handler function using Money.Net data, see `realtime`. For custom event handlers, see “Writing and Running Custom Event Handler Functions” on page 1-26. |

## See Also

`close` | `moneynet` | `news` | `realtime` | `timeseries`

## More About

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-5
- “Retrieve Money.Net News Stories” on page 5-7
- “Writing and Running Custom Event Handler Functions” on page 1-26

## External Websites

- Money.Net API Documentation
- Money.Net Help

# Elektron Topics


- “Decide to Buy Shares Using Elektron Current Data” on page 6-2
- “Decide to Buy Shares Using Elektron Real-Time Data” on page 6-4

# Decide to Buy Shares Using Elektron Current Data

This example shows how to connect to Elektron from Refinitiv and trigger a buy decision for a single RIC using the current Elektron last trade price.

To access the code for this example, enter `edit ElektronWorkflow.m`.

### Create Elektron Connection

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

### Retrieve Current Data for Single Security

Retrieve current data for the IBM security.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type of the Elektron field
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

    FieldId    DataType        Name                Value
    _______    ________    ____________    ____________________________
```

```
    [    1]    [18]        "PROD_PERM"     ["62"                   ]
    [    2]    [18]        "RDNDISPLAY"    ["64"                   ]
    [    3]    [31]        "DSPLY_NAME"    ["DELAYED-15INTL B"     ]
...
```

### Make Investment Decision to Buy Shares

Assume a price threshold of $175. Determine if the last trade price is less than $175. Set the buy indicator `buynow` to `true` when the threshold is met. Find the Elektron last trade price by the field identifier `6`.

```
for i = 1:height(d)
    field = cell2mat(d.FieldId(i)); % Convert cell array value to numeric
    value = cell2mat(d.Value(i));
    if (field == 6)                 % Find last trade price
        if (value < 175)            % Trigger buy price threshold
            buynow = true;
        end
    end
end
```

Use the buy indicator to create a buy order for IBM shares in the trading system of your choice.

### Close Elektron Connection

```
close(c)
```

## See Also

`close` | `elektron` | `getdata`

## Related Examples

- “Decide to Buy Shares Using Elektron Real-Time Data” on page 6-4

## External Websites

- Elektron from Refinitiv

# Decide to Buy Shares Using Elektron Real-Time Data

This example shows how to connect to Elektron from Refinitiv and trigger a buy decision for multiple RICs using the real-time Elektron last trade price.

The example uses the sample event handler function `elektronExampleListener` to retrieve real-time data for multiple securities in corresponding MATLAB workspace variables. Use this event handler function or write a custom event handler function. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

To access the code for this example, enter `edit ElektronWorkflow.m`.

### Create Elektron Connection

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

### Retrieve Real-Time Data for Multiple Securities

Retrieve real-time market data for the IBM and Microsoft securities using the Elektron connection. Use the example event handler function `elektronExampleListener`. The `realtime` function returns the variable `reqid` as a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
seclist = {'IBM.N','MSFT.O'};
eventhandler = @(~,ev)elektronExampleListener(ev);
reqid = realtime(c,seclist,eventhandler)

reqid =

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The cell arrays `IBM` and `MSFT` appear in the MATLAB workspace. Each cell array contains the same four columns. The columns are:

- Elektron field identifier
- Elektron field name
- Elektron field data type
- Elektron field real-time data value

### Make Investment Decisions to Buy Shares

Assume a price threshold of $175 for the IBM security. Determine if the last trade price is less than $175. Set the buy indicator `buynow` to `true` when the threshold is met. The first column in the cell array `IBM` contains the field identifier. The fourth column in the cell array contains the field value. Find the Elektron last trade price by the field identifier `6`.

```
for i = 1:length(IBM)
    if (IBM{i,1} == 6)      % Find last trade price
        if (IBM{i,4} < 175) % Trigger buy price threshold
            buynow = true;
        end
    end
end
```

Use the buy indicator to create a buy order for IBM shares in the trading system of your choice.

Assume a price threshold of $75 for the Microsoft security. Determine if the last trade price is less than $75. Set the buy indicator `buynow` to `true` when the threshold is met. The first column in the cell array `MSFT` contains the field identifier. The fourth column in the cell array contains the field value. Find the Elektron last trade price by the field identifier `6`.

```
for i = 1:length(MSFT)
    if (MSFT{i,1} == 6)     % Find last trade price
        if (MSFT{i,4} < 75) % Trigger buy price threshold
            buynow = true;
        end
    end
end
```

Use the buy indicator to create a buy order for Microsoft shares in the trading system of your choice.

### Stop Real-Time Data Subscription

```
delete(reqid.Listener)
```

### Close Elektron Connection

```
close(c)
```

## See Also

`close` | `elektron` | `realtime`

## More About

- “Decide to Buy Shares Using Elektron Current Data” on page 6-2
- “Writing and Running Custom Event Handler Functions” on page 1-26

## External Websites

- Elektron from Refinitiv

# Twitter Topics


- “Conduct Sentiment Analysis Using Historical Tweets” on page 7-2
- “Tweet Based on Retrieved Twitter Data” on page 7-6

# Conduct Sentiment Analysis Using Historical Tweets

This example shows how to search and retrieve all available Tweets in the last 7 days and import them into MATLAB. After importing the data, you can conduct sentiment analysis. This analysis enables you to determine subjective information, such as moods, opinions, or emotional reactions, from text data. This example searches for positive and negative moods regarding the financial services industry.

To run this example, you need Twitter credentials. To obtain these credentials, you must first log in to your Twitter account. Then, fill out the form in Create an application.

To access the example code, enter `edit TwitterExample.m` at the command line.

### Connect to Twitter

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode

ans =

    OK
```

### Retrieve Latest Tweets

Search for the latest 100 Tweets about the financial services industry using the Twitter connection object. Use the search term `financial services`. Import Tweet® data into the MATLAB workspace.

```
tweetquery = 'financial services';
s = search(c,tweetquery,'count',100);
statuses = s.Body.Data.statuses;
pause(2)
```

`statuses` contains the Tweet data as a cell array of 100 structures. Each structure contains a field for the Tweet text, and the remaining fields contain other information about the Tweet.

Search and retrieve the next 100 Tweets that have occurred since the previous request.

```
sRefresh = search(c,tweetquery,'count',100, ...
    'since_id',s.Body.Data.search_metadata.max_id_str);
statuses = [statuses;sRefresh.Body.Data.statuses];
```

`statuses` contains the latest 100 Tweets in addition to the previous 100 Tweets.

### Retrieve All Available Tweets

Retrieve all available Tweets about the financial services industry using a `while` loop. Check for available data using the `isfield` function and the structure field `next_results`.

```
while isfield(s.Body.Data.search_metadata,'next_results')
    % Convert results to string
    nextresults = string(s.Body.Data.search_metadata.next_results);
    % Extract maximum Tweet identifier
    max_id = extractBetween(nextresults,"max_id=","&");
    % Convert maximum Tweet identifier to a character vector
    cmax_id = char(max_id);
    % Search for Tweets
    s = search(c,tweetquery,'count',100,'max_id',cmax_id);
    % Retrieve Tweet text for each Tweet
    statuses = [statuses;s.Body.Data.statuses];
end
```

Retrieve the creation time and text of each Tweet. Retrieve the creation time for unstructured data by accessing it in a cell array of structures. For structured data, access the creation time by transposing the field in the structure array.

```
if iscell(statuses)
  % Unstructured data
    numTweets = length(statuses);             % Determine total number of Tweets
    tweetTimes = cell(numTweets,1);           % Allocate space for Tweet times and Tweet text
    tweetTexts = tweetTimes;
    for i = 1:numTweets
      tweetTimes{i} = statuses{i}.created_at; % Retrieve the time each Tweet was created
      tweetTexts{i} = statuses{i}.text;       % Retrieve the text of each Tweet
    end
else
    % Structured data
    tweetTimes = {statuses.created_at}';
    tweetTexts = {statuses.text}';
end
```

`tweetTimes` contains the creation time for each Tweet. `tweetTexts` contains the text for each Tweet.

Create the timetable `tweets` for all Tweets by using the text and creation time of each Tweet.

```
tweets = timetable(tweetTexts,'RowTimes', ...
    datetime(tweetTimes,'Format','eee MMM dd HH:mm:ss +SSSS yyyy'));
```

### Conduct Sentiment Analysis on Tweets

Create a glossary of words that are associated with positive sentiment.

```
poskeywords = {'happy','great','good', ...
    'fast','optimized','nice','interesting','amazing','top','award', ...
    'winner','wins','cool','thanks','useful'};
```

`poskeywords` is a cell array of character vectors. Each character vector is a word that represents an instance of positive sentiment.

Search each Tweet for words in the positive sentiment glossary. Determine the total number of Tweets that contain a positive sentiment. Out of the total number of positive Tweets, determine the total number of Retweets.

```
% Determine the total number of Tweets
numTweets = height(tweets);
```

```
% Determine the positive Tweets
numPosTweets = 0;
numPosRTs = 0;
for i = 1:numTweets
    % Compare Tweet to positive sentiment glossary
    dJobs = contains(tweets.tweetTexts{i},poskeywords,'IgnoreCase',true);
    if dJobs
        % Increase total count of Tweets with positive sentiment by one
        numPosTweets = numPosTweets + 1;
        % Determine if positive Tweet is a Retweet
        RTs = strncmp('RT @',tweets.tweetTexts{i},4);
        if RTs
            % Increase total count of positive Retweets by one
            numPosRTs = numPosRTs + 1;
        end
    end
end
```

`numPosTweets` contains the total number of Tweets with positive sentiment.

`numPosRTs` contains the total number of Retweets with positive sentiment.

Create a glossary of words that are associated with negative sentiment.

```
negkeywords = {'sad','poor','bad','slow','weaken','mean','boring', ...
    'ordinary','bottom','loss','loser','loses','uncool', ...
    'criticism','useless'};
```

`negkeywords` is a cell array of character vectors. Each character vector is a word that represents an instance of negative sentiment.

Search each Tweet for words in the negative sentiment glossary. Determine the total number of Tweets that contain a negative sentiment. Out of the total number of negative Tweets, determine the total number of Retweets.

```
% Determine the negative Tweets
numNegTweets = 0;
numNegRTs = 0;
for i = 1:numTweets
    % Compare Tweet to negative sentiment glossary
    dJobs = contains(tweets.tweetTexts{i},negkeywords,'IgnoreCase',true);
    if dJobs
        % Increase total count of Tweets with negative sentiment by one
        numNegTweets = numNegTweets + 1;
        % Determine if negative Tweet is a Retweet
        RTs = strncmp('RT @',tweets.tweetTexts{i},4);
        if RTs
            numNegRTs = numNegRTs + 1;
        end
    end
end
```

`numNegTweets` contains the total number of Tweets with negative sentiment.

`numNegRTs` contains the total number of Retweets with negative sentiment.

### Display Sentiment Analysis Results

Create a table with columns that contain:

- Number of Tweets
- Number of Tweets with positive sentiment
- Number of positive Retweets
- Number of Tweets with negative sentiment
- Number of negative Retweets

```
matlabTweetTable = table(numTweets,numPosTweets,numPosRTs,numNegTweets,numNegRTs, ...
    'VariableNames',{'Number_of_Tweets','Positive_Tweets','Positive_Retweets', ...
    'Negative_Tweets','Negative_Retweets'});
```

Display the table of Tweet data.

```
matlabTweetTable
```

```
matlabTweetTable =

  1×5 table

    Number_of_Tweets    Positive_Tweets    Positive_Retweets    Negative_Tweets    Negative_Retweets
    ________________    _______________    _________________    _______________    _________________

    11465               688                238                  201                96
```

Out of 11,465 total Tweets about the financial services industry in the last 7 days, 688 Tweets have positive sentiment and 201 Tweets have negative sentiment. Out of the positive Tweets, 238 Tweets are Retweets. Out of the negative Tweets, 96 are Retweets.

## See Also

### Functions

`search`

### Objects

`twitter`

## More About

- “Tweet Based on Retrieved Twitter Data” on page 7-6

## External Websites

- Twitter REST API Endpoint Reference Documentation

# Tweet Based on Retrieved Twitter Data

This example shows how to retrieve the number of followers for a Twitter account and Tweet about achieving a specific follower count. You can adapt this example to retrieve data from other Twitter REST API endpoints, such as collections, lists, and so on.

To run this example, you need Twitter credentials. To obtain these credentials, you must first log in to your Twitter account. Then, fill out the form in Create an application.

### Connect to Twitter

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

### Retrieve Number of Followers

Set the Twitter base URL to access the `GET followers/ids` REST API endpoint. Search for a specific Twitter account using the Twitter connection object, base URL, and screen name. (The screen name in this example does not represent real Twitter data.)

```
baseurl = 'https://api.twitter.com/1.1/followers/ids.json';
sname = 'screenname';
d = getdata(c,baseurl,'screen_name',sname)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful request.

Determine the number of followers for the specified account.

```
numfollowers = length(d.Body.Data.ids)
```

```
numfollowers =

    44
```

This account has 44 followers.

**Post Tweet**

Create the character vector `tweetString`, which specifies the Tweet to post. If the number of followers is greater than 25, then the Tweet indicates the screen name has more than 25 followers. Otherwise, it indicates that the screen name needs more followers.

```
if numfollowers > 25
    tweetString = [sname ' has more than 25 followers!'];
else
    tweetString = [sname ' needs more followers!'];
end
```

Set the Twitter base URL to access the `POST statuses/update` REST API endpoint.

```
baseurl = 'https://api.twitter.com/1.1/statuses/update.json';
```

Tweet about the number of followers using the Twitter connection object, base URL, and `tweetString`.

```
d = postdata(c,baseurl,'status',tweetString)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×22 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful request.

## See Also

**Functions**
`getdata` | `postdata`

**Objects**
`twitter`

## More About

- “Conduct Sentiment Analysis Using Historical Tweets” on page 7-2

## External Websites

- Twitter REST API Endpoint Reference Documentation

# Tick History from Refinitiv Topics


- “Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv” on page 8-2
- “Decide to Sell Shares with Historical Data Using Tick History from Refinitiv” on page 8-4

# Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv

This example shows how to connect to Tick History from Refinitiv and trigger a buy decision for a single Reuters Instrument Code (RIC) using the trade price.

Create a Tick History from Refinitiv connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve intraday data for the IBM security. Using the `timeseries` function, retrieve the trade price from November 6, 2017 through November 7, 2017.

```
sec = ["IBM.N","Ric"];
fields = ["Trade - Price"];
startdate = datetime('11/06/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/07/2017','InputFormat','MM/dd/yyyy');

d = timeseries(c,sec,fields,startdate,enddate);
```

Display the first three rows of intraday data.

```
head(d,3)
```

```
ans =

  3×5 timetable

           Time             x_RIC         Domain        GMTOffset     Type       Price
    ____________________    _____    ______________    _________    _______    ________

    06-Nov-2017 14:30:10    'IBM.N'    'Market Price'      '-5'       'Trade'    '151.68'
    06-Nov-2017 14:30:10    'IBM.N'    'Market Price'      '-5'       'Trade'    '151.66'
    06-Nov-2017 14:30:10    'IBM.N'    'Market Price'      '-5'       'Trade'    '151.73'
```

`d` is a timetable that contains these variables:

- Transaction date and time
- RIC
- Domain
- GMT time zone offset
- Transaction type
- Price

Assume a price threshold of $160. Determine if the trade price is less than $160. Set the buy indicator `buynow` to `true` when the threshold is met.

```
value = str2double(d.Price);
buynow = (value < 160);
```

Use the buy indicator to create a buy order of IBM shares in the trading system of your choice.

## See Also

`timeseries` | `trth`

## More About

- “Decide to Sell Shares with Historical Data Using Tick History from Refinitiv” on page 8-4

## External Websites

- Refinitiv REST API Documentation

# Decide to Sell Shares with Historical Data Using Tick History from Refinitiv

This example shows how to connect to Tick History from Refinitiv and trigger a sell decision for a single Reuters Instrument Code (RIC) using the closing price.

Create a Tick History from Refinitiv connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve historical data for the IBM security. Using the `history` function, retrieve the closing price from November 6, 2017 through November 7, 2017.

```
sec = ["IBM.N","Ric"];
fields = ["Last"];
startdate = datetime('11/06/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/07/2017','InputFormat','MM/dd/yyyy');

d = history(c,sec,fields,startdate,enddate);
```

```
  2×1 timetable

      Time        Last
    __________    ______

    2017/11/06    150.84
    2017/11/07    151.35
```

Assume a price threshold of $150. Determine if the closing price is greater than $150. Set the sell indicator `sellnow` to `true` when the threshold is met.

```
sellnow = (d.Last > 150);
```

Use the sell indicator to create a sell order of IBM shares in the trading system of your choice.

## See Also

`history` | `trth`

## More About

- “Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv” on page 8-2

## External Websites

- Refinitiv REST API Documentation

# Quandl Topics

# Access Quandl Error Messages

When you request historical data from Quandl, sometimes the request returns an error instead of the historical data. Use this workflow to access Quandl error messages.

The `history` function returns errors in the `matlab.net.http.ResponseMessage` object. For example, suppose that you enter an invalid security name for the `s` input argument. The resulting output has this form:

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 404 Not Found'
    StatusCode: NotFound
        Header: [1×19 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

Access the `Body` property using dot notation.

```
d.Body
```

```
ans =

  MessageBody with properties:

              Data: [1×1 struct]
           Payload: []
       ContentType: [1×1 matlab.net.http.MediaType]
    ContentCoding: [0×0 string]
```

To view the text of the error message, access the nested structure `quandl_error` stored in the `Data` property.

```
d.Body.Data.quandl_error
```

```
ans =

  struct with fields:

       code: 'QECx02'
    message: 'You have submitted an incorrect Quandl code. Please check your Quandl codes and try
```

Each error has a code and message associated with it. To view the code, access the `code` field. To view the error message text, access the `message` field. For example:

```
d.Body.Data.quandl_error.message
```

```
ans =

    'You have submitted an incorrect Quandl code. Please check your Quandl codes and try again.'
```

Refer to the error message to fix your code.

## See Also

`history` | `quandl`

## More About

- “Retrieve Historical Data Using Quandl” on page 1-21
- “Access Data in Nested Structures”

## External Websites

- Quandl

# Datastream Web Services Topics


- “Retrieve Datastream Web Services Historical Data” on page 10-2
- “Access Datastream Web Services Error Messages” on page 10-4

# Retrieve Datastream Web Services Historical Data

This example shows how to retrieve historical data from Datastream Web Services from Refinitiv. You must have Datastream Web Services credentials. For credentials, contact Datastream Web Services from Refinitiv.

### Create Datastream Web Services Connection

Create a Datastream Web Services connection using your user name and password. `c` is the `datastreamws` connection object.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

### Retrieve Monthly Historical Data

Adjust the display format to display currency.

```
format bank
```

Retrieve and display historical end-of-day price data from January 1, 2017 through December 31, 2017. Specify the `VOD` security and these fields:

- Opening price
- High price
- Last closing price

Specify a monthly period. `d` is a timetable with the date in the first variable and the fields in the subsequent variables.

```
sec = "VOD";
fields = ["PO";"PH";"P"];
startdate = datetime('01-01-2017','InputFormat','MM-dd-yyyy');
enddate = datetime('12-31-2017','InputFormat','MM-dd-yyyy');
period = 'M';
d = history(c,sec,fields,startdate,enddate,period)
```

```
d =

  12×3 timetable

          Time              PO        PH        P
    ____________________    ______    ______    ______

    01-Jan-2017 00:00:00       NaN       NaN    199.85
    01-Feb-2017 00:00:00    196.85    197.25    193.00
    01-Mar-2017 00:00:00    201.80    202.55    202.55
    01-Apr-2017 00:00:00    209.00    209.10    206.65
    01-May-2017 00:00:00       NaN       NaN    199.05
    01-Jun-2017 00:00:00    231.65    233.90    229.40
    01-Jul-2017 00:00:00    217.65    219.20    218.70
    01-Aug-2017 00:00:00    223.15    223.60    221.65
    01-Sep-2017 00:00:00    221.25    221.95    219.50
    01-Oct-2017 00:00:00    209.35    211.60    210.50
    01-Nov-2017 00:00:00    217.00    222.30    218.95
    01-Dec-2017 00:00:00    224.15    230.65    224.00
```

### Retrieve Quarterly Historical Data

Retrieve and display historical end-of-day price data from January 1, 2017 through December 31, 2017. Specify the `VOD` security and these fields:

- Opening price
- High price
- Last closing price

Specify a quarterly period. `d` is a timetable with the date in the first variable and the fields in the subsequent variables.

```
sec = "VOD";
fields = ["PO";"PH";"P"];
startdate = datetime('01-01-2017','InputFormat','MM-dd-yyyy');
enddate = datetime('12-31-2017','InputFormat','MM-dd-yyyy');
period = 'Q';
d = history(c,sec,fields,startdate,enddate,period)
```

```
d =

  4×3 timetable

           Time              PO        PH        P
    ____________________    ______    ______    ______

    01-Jan-2017 00:00:00       NaN       NaN    199.85
    01-Apr-2017 00:00:00    209.00    209.10    206.65
    01-Jul-2017 00:00:00    217.65    219.20    218.70
    01-Oct-2017 00:00:00    209.35    211.60    210.50
```

Use the monthly and quarterly prices for each field to make investment decisions for the `VOD` security.

## See Also

`datastreamws` | `history`

## More About

- “Access Datastream Web Services Error Messages” on page 10-4

## External Websites

- Datastream Web Services from Refinitiv REST Service

# Access Datastream Web Services Error Messages

When you make a historical data request from Datastream Web Services from Refinitiv, sometimes the request returns an error instead of data. Use this workflow to access Datastream Web Services error messages.

The `history` function returns errors in the `matlab.net.http.ResponseMessage` object. For example, suppose that you enter an invalid security name for the `sec` input argument. The resulting output has this form:

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×6 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

Access the `Body` property using dot notation.

```
d.Body
```

```
ans =

  MessageBody with properties:

              Data: [1×1 struct]
           Payload: []
       ContentType: [1×1 matlab.net.http.MediaType]
    ContentCoding: [0×0 string]
```

To access the text of the error message, access the nested structure `DataResponse` stored in the `Data` property.

```
d.Body.Data.DataResponse
```

```
ans =

  struct with fields:

    AdditionalResponses: []
          DataTypeNames: []
         DataTypeValues: [3×1 struct]
                  Dates: []
            SymbolNames: []
                    Tag: ''
```

Then, access the `SymbolValues` field in the `DataTypeValues` structure array.

```
d.Body.Data.DataResponse.DataTypeValues(1).SymbolValues
```

```
ans =

  struct with fields:

    Currency: []
```

```
      Symbol: 'YYY'
        Type: 0
       Value: '$$ER: E100,INVALID CODE OR EXPRESSION ENTERED'
```

Fix your code based on the error message in the `Value` field.

## See Also

`datastreamws` | `history`

## More About

- “Retrieve Datastream Web Services Historical Data” on page 10-2
- “Access Data in Nested Structures”

## External Websites

- Datastream Web Services from Refinitiv REST Service

# IHS Markit Topics


- “Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
- “IHS Markit Error Messages” on page 11-4

# Retrieve Factor Rank Data for Portfolio Selection

This example shows how to retrieve ranking data from IHS Markit for use in portfolio selection or an existing model. Retrieve percentile rank data for ticker security identifiers of a factor code. Then, use the rank information for portfolio selection or further analysis in an existing model. The example assumes that you have IHS Markit credentials. For credentials, see the IHS Markit website.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve signal information for the last 10 days using the IHS Markit connection. Specify the `ABR` factor code and `US Total Cap` universe. Also, specify the ticker security type and percentile data format. The percentile format provides factor ranking data. `d` is a table that contains signal information and the `date` and `data` variables.

```
code = 'ABR';
universeid = 'US Total Cap';
startdate = datetime('today')-10;
enddate = datetime('today');
identifier = 'ticker';
datatype = 'percentile';
d = signals(c,code,universeid,startdate,enddate,identifier,datatype);
```

Access the first few rows of ranking data for the first day in the date range by using the `data` variable.

```
data = d.data{1};
head(data)
```

```
ans =

  8×2 table

    ticker    value
    ______    _____

    'SVU'       1
    'LBY'       1
    'TLRY'      1
    'WIFI'      1
    'TCS'       1
    'AOBC'      1
    'TTD'       1
    'ZOES'      1
```

The variables of the resulting table are `ticker` and `value`. The `ticker` variable contains the ticker security identifiers. The `value` variable contains the factor ranking data.

Find all ticker security identifiers in `data` that have the most attractive value using rank values `1` through `10`. Create a table to store the rank values and perform an inner join to retrieve the most attractive securities. Display the last few attractive securities.

```
value = 1:10; % Define array of ranks 1 through 10
T = table(value','VariableNames',{'value'}); % Create table of the ranks in one variable
```

```
securities = innerjoin(data,T); % Perform inner join to find securities within the ranks
tail(securities)
```

```
ans =

  8×2 table

    ticker     value
    _______    _____

    'CDPYF'     10
    'CNXN'      10
    'DRNA'      10
    'PSX'       10
    'BRC'       10
    'ICHR'      10
    'MNLO'      10
    'MBI'       10
```

Use the factor rank data in your portfolio selection process or further analysis in your existing model.

## See Also

`ihsmarkitrs` | `signals`

## More About

- “IHS Markit Error Messages” on page 11-4

## External Websites

- IHS Markit

# IHS Markit Error Messages

This table describes how to address common errors you can encounter while working with IHS Markit.

| Error Message | Probable Causes | Resolution |
|---|---|---|
| Unknown filter. | The specified factor code is invalid. | Specify a valid factor code. |
| Unknown Universe Name. | The specified universe name is invalid. | Specify a valid universe name. |
| Governor exceeded. More than 30 between *mm/DD/YYYY HH:MM:ss* and *mm/DD/YYYY HH:MM:ss* | For the specified date range, the returned data is too large. | Specify a shorter date range. |

## See Also

`factors` | `ihsmarkitrs` | `security` | `signals` | `universes`

## More About

- “Retrieve Factor Rank Data for Portfolio Selection” on page 11-2

## External Websites

- IHS Markit
- IHS Markit Research Signals REST Documentation

# Functions

# blp

Bloomberg Desktop connection V3

## Description

The `blp` function creates a `blp` object. The `blp` object represents a Bloomberg Desktop connection.

Other functions connect to different Bloomberg services: Bloomberg Server (`blpsrv`), Bloomberg B-PIPE (`bpipe`), and Bloomberg Data License (`bdl`). For details about these services, see “Comparing Bloomberg Connections” on page 2-3.

For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing `blp`. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

## Creation

### Syntax

```
c = blp
c = blp(port,ip,timeout)
```

#### Description

`c = blp` creates a Bloomberg connection object that contains the Bloomberg Desktop connection. You need a Bloomberg Desktop software license for the machine running the Datafeed Toolbox and MATLAB.

`c = blp(port,ip,timeout)` sets the port and timeout properties, and uses the IP address of the local machine running Bloomberg to create a Bloomberg connection.

**Caution:** To refer to a Bloomberg connection in other functions, use the connection object created by the `blp` function. Otherwise, using `blp` as an input argument opens multiple Bloomberg connections, causing unexpected behavior and exhausting memory resources.

#### Input Arguments

**`ip` — IP address**
`[ ]` (default) | character vector | string scalar

IP address that identifies the local machine running Bloomberg, specified as a character vector or string scalar.

Example: `'localhost'`

Data Types: `char` | `string`

## Properties

### `Session` — Bloomberg V3 session

Bloomberg V3 API Session object

This property is read-only.

Bloomberg V3 session, specified as a Bloomberg V3 API Session object.

Example: `[1x1 com.bloomberglp.blpapi.Session]`

### `Port` — Port number of local machine

`[]` (default) | numeric scalar

Port number of the local machine running Bloomberg, specified as a numeric scalar.

Example: `8194`

Data Types: `double`

### `IPAddress` — IP address of local machine

`[]` (default) | character vector

IP address of the local machine running Bloomberg, specified as a character vector.

The `blp` function sets this property using the `ip` input argument.

Example: `'localhost'`

Data Types: `char`

### `TimeOut` — Timeout

numeric scalar

Timeout specifying the time in milliseconds that MATLAB attempts to connect to Bloomberg Desktop before timing out, specified as a numeric scalar.

Example: `10000`

Data Types: `double`

### `DatetimeType` — Date and time data type

`''` (default) | `'datetime'`

Date and time data type, specified as one of these values.

| Value | Description |
|---|---|
| `''` (default) | Return date and time values as MATLAB date numbers. |
| `'datetime'` | Return date and time values as a `datetime` array. |

You can specify these values using a character vector or string (for example, `"datetime"`).

When you create a `blp` object, the `blp` function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DatetimeType = 'datetime';
```

Then, you can use these supported functions:

- `getbulkdata`
- `getdata`
- `history`
- `tahistory`
- `timeseries`

> **Note** If the `DataReturnFormat` property value is `'table'` and the `DatetimeType` property value is `'datetime'`, then the returned data is a table that contains date and time values as a `datetime` array. If the `DataReturnFormat` property value is an empty character vector, then setting the `DatetimeType` property to `'datetime'` returns date and time values for aggregated ticks and historical requests as MATLAB date numbers.

**`DataReturnFormat` — Data return format**
`'cell'` | `'structure'` | `'table'` | `'timetable'`

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| `'cell'` | `cell array` |
| `'table'` | `table` |
| `'timetable'` | `timetable` |
| `'structure'` | `structure` |

**Note** The default data type of the returned data depends on the executed function. To specify the default data type, set the `DataReturnFormat` property to `''`. For default data types, see the supported function list.

You can specify these values using a character vector or string (for example, `"table"`).

When you create a `blp` object, the `blp` function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'structure';
```

Then, you can use these supported functions.

| Supported Function | Valid Data Types for Returned Data |
|---|---|
| category | • `cell array` (default)<br>• `structure`<br>• `table` |
| eqs | • `cell array` (default)<br>• `structure`<br>• `table` |
| fieldinfo | • `cell array` (default)<br>• `structure`<br>• `table` |
| fieldsearch | • `cell array` (default)<br>• `structure`<br>• `table` |
| lookup | • `structure` (default)<br>• `table` |
| portfolio | • `structure` (default)<br>• `table` |
| getbulkdata | • `structure` (default)<br>• `table`<br>• `timetable` |
| getdata | • `structure` (default)<br>• `table`<br>• `timetable` |
| history | • numeric array (default)<br>• `table`<br>• `timetable` |
| tahistory | • `structure` (default)<br>• `table`<br>• `timetable` |
| timeseries | • `cell array` (default for raw tick data)<br>• numeric array (default for interval tick data)<br>• `table`<br>• `timetable` |

**Note** Regardless of the `DatetimeType` property value, if the `DataReturnFormat` property value is `'timetable'`, then the `getdata` and `getbulkdata` functions return a table that contains date and time values as `datetime` arrays.

## Object Functions

## Bloomberg Desktop Connection

| | |
|---|---|
| close | Close Bloomberg connection V3 |
| get | Properties of Bloomberg connection V3 |
| isconnection | Determine Bloomberg connection V3 |

## Bloomberg Desktop Data Retrieval

| | |
|---|---|
| eqs | Equity screening data for Bloomberg connection V3 |
| getbulkdata | Bulk data with header information for Bloomberg connection V3 |
| getdata | Current data for Bloomberg connection V3 |
| history | Historical data for Bloomberg connection V3 |
| portfolio | Current portfolio data for Bloomberg connection V3 |
| realtime | Real-time data for Bloomberg connection V3 |
| stop | Unsubscribe real-time requests for Bloomberg connection V3 |
| tahistory | Historical technical analysis for Bloomberg connection V3 |
| timeseries | Intraday tick data for Bloomberg connection V3 |

## Bloomberg Desktop Data Information

| | |
|---|---|
| category | Field category search for Bloomberg connection V3 |
| fieldinfo | Field information for Bloomberg connection V3 |
| fieldsearch | Field search for Bloomberg connection V3 |
| lookup | Find information about securities for Bloomberg connection V3 |

## Examples

### Connect to Bloomberg Desktop

First, create a Bloomberg® connection, and then retrieve current data for a security.

Create a connection to the Bloomberg Desktop.

```
c = blp
```

```
c =
  blp with properties:

            Session: [1×1 com.bloomberglp.blpapi.Session]
          IPAddress: 'localhost'
               Port: 8194
            TimeOut: 0
       DatetimeType: ''
    DataReturnFormat: ''
```

`c` is a Bloomberg connection object with these properties:

- Bloomberg V3 API Session object
- IP address of the local machine
- Port number of the local machine
- Number in milliseconds specifying how long MATLAB attempts to connect to Bloomberg Desktop before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft®.

```
format bank % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)
```

```
d = struct with fields:
    LAST_PRICE: 72.28
          OPEN: 71.61


sec = 1×1 cell array
    {'MSFT US Equity'}
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg Desktop connection.

```
close(c)
```

### Connect to Bloomberg Desktop with Timeout

First, create a Bloomberg® connection with a timeout value, and then retrieve current data for a security.

Create a connection to the Bloomberg Desktop using the default port and IP address. Specify a timeout value of 10,000 milliseconds.

```
c = blp([],[],10000)
```

```
c =
  blp with properties:

            Session: [1×1 com.bloomberglp.blpapi.Session]
          IPAddress: 'localhost'
               Port: 8194
            TimeOut: 10000
       DatetimeType: ''
    DataReturnFormat: ''
```

The `blp` function creates a Bloomberg connection object `c` with these properties:

- Bloomberg V3 API Session object
- IP address of the local machine
- Port number of the local machine
- Number of milliseconds specifying how long MATLAB® attempts to connect to Bloomberg Desktop before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft®.

```
format bank % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)
```

```
d = struct with fields:
    LAST_PRICE: 71.83
          OPEN: 71.61


sec = 1×1 cell array
    {'MSFT US Equity'}
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg Desktop connection.

```
close(c)
```

## See Also

### Topics

“Connect to Bloomberg” on page 3-2
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-3
“Workflow for Bloomberg” on page 3-19

### Introduced in R2010a

# blpsrv

Bloomberg Server connection V3

# Description

The `blpsrv` function creates a `blpsrv` object. The `blpsrv` object represents a Bloomberg Server connection.

Other functions connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg B-PIPE (`bpipe`), and Bloomberg Data License (`bdl`). For details about these services, see “Comparing Bloomberg Connections” on page 2-3.

For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing `blpsrv`. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

# Creation

## Syntax

```
c = blpsrv(uuid,ipaddress)
c = blpsrv(uuid,ipaddress,port)
c = blpsrv(uuid,ipaddress,port,timeout)
```

### Description

`c = blpsrv(uuid,ipaddress)` creates a Bloomberg Server connection object `c` to the Bloomberg Server running on another machine, and sets the Uuid and IPAddress properties. You need a Bloomberg Server software license for the machine running the Bloomberg Server.

`c = blpsrv(uuid,ipaddress,port)` also sets the port property.

`c = blpsrv(uuid,ipaddress,port,timeout)` also sets the timeout property.

**Caution:** To refer to a Bloomberg connection in other functions, use the connection object created by the `blpsrv` function. Otherwise, using `blpsrv` as an input argument opens multiple Bloomberg connections, causing unexpected behavior and exhausting memory resources.

## Properties

### `Uuid` — Bloomberg user identity UUID

numeric scalar

Bloomberg user identity UUID, specified as a numeric scalar. To find your UUID, enter `IAM` in the Bloomberg terminal and press **GO**.

Example: 12345678

Data Types: `double`

**`User` — Bloomberg user**
Bloomberg user identity object

This property is read-only.

Bloomberg user, specified as a Bloomberg user identity object.

Example: `[1x1 com.bloomberglp.blpapi.impl.aT]`

**`Userip` — IP address of the machine running MATLAB**
character vector

This property is read-only.

IP address of the machine running MATLAB, specified as a character vector.

Example: `'111.11.11.111'`

Data Types: `char`

**`Session` — Bloomberg V3 session**
Bloomberg V3 API Session object

This property is read-only.

Bloomberg V3 session, specified as a Bloomberg V3 API Session object.

Example: `[1x1 com.bloomberglp.blpapi.Session]`

**`IPAddress` — Bloomberg Server IP address**
character vector | string scalar

Bloomberg Server IP address, specified as a character vector or string scalar that identifies the machine running the Bloomberg Server.

Example: `'111.11.11.111'`

Data Types: `char` | `string`

**`Port` — Port number**
numeric scalar

Port number, specified as a numeric scalar that identifies the port number of the machine running the Bloomberg Server.

Example: `8194`

Data Types: `double`

**`TimeOut` — Timeout**
numeric scalar

Timeout specifying the time in milliseconds that MATLAB attempts to connect to the machine running the Bloomberg Server before timing out, specified as a numeric scalar.

Example: `10`

Data Types: `double`

### `DatetimeType` — Date and time data type

`''` (default) | `'datetime'`

Date and time data type, specified as one of these values.

| Value | Description |
|---|---|
| `''` (default) | Return date and time values as MATLAB date numbers. |
| `'datetime'` | Return date and time values as a `datetime` array. |

You can specify these values using a character vector or string (for example, `"datetime"`).

When you create a `blpsrv` object, the `blpsrv` function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DatetimeType = 'datetime';
```

Then, you can use these supported functions:

- `getbulkdata`
- `getdata`
- `history`
- `tahistory`
- `timeseries`

> **Note** If the `DataReturnFormat` property value is `'table'` and the `DatetimeType` property value is `'datetime'`, then the returned data is a table that contains date and time values as a `datetime` array. If the `DataReturnFormat` property value is an empty character vector, then setting the `DatetimeType` property to `'datetime'` returns date and time values for aggregated ticks and historical requests as MATLAB date numbers.

### `DataReturnFormat` — Data return format

`'cell'` | `'structure'` | `'table'` | `'timetable'`

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| `'cell'` | `cell array` |
| `'table'` | `table` |
| `'timetable'` | `timetable` |

| Value | Data Type of Returned Data |
|---|---|
| `'structure'` | `structure` |

**Note** The default data type of the returned data depends on the executed function. To specify the default data type, set the `DataReturnFormat` property to `''`. For default data types, see the supported function list.

You can specify these values using a character vector or string (for example, `"table"`).

When you create a `blpsrv` object, the `blpsrv` function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'structure';
```

Then, you can use these supported functions.

| Supported Function | Valid Data Types for Returned Data |
|---|---|
| `category` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `eqs` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `fieldinfo` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `fieldsearch` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `lookup` | • `structure` (default)<br>• `table` |
| `portfolio` | • `structure` (default)<br>• `table` |
| `getbulkdata` | • `structure` (default)<br>• `table`<br>• `timetable` |

| Supported Function | Valid Data Types for Returned Data |
|---|---|
| `getdata` | • `structure` (default)<br>• `table`<br>• `timetable` |
| `history` | • numeric array (default)<br>• `table`<br>• `timetable` |
| `tahistory` | • `structure` (default)<br>• `table`<br>• `timetable` |
| `timeseries` | • `cell array` (default for raw tick data)<br>• numeric array (default for interval tick data)<br>• `table`<br>• `timetable` |

**Note** Regardless of the `DatetimeType` property value, if the `DataReturnFormat` property value is `'timetable'`, then the `getdata` and `getbulkdata` functions return a table that contains date and time values as `datetime` arrays.

## Object Functions

## Bloomberg Server Connection

close Close Bloomberg connection V3
get Properties of Bloomberg connection V3
isconnection Determine Bloomberg connection V3

## Bloomberg Server Data Retrieval

eqs Equity screening data for Bloomberg connection V3
getbulkdata Bulk data with header information for Bloomberg connection V3
getdata Current data for Bloomberg connection V3
history Historical data for Bloomberg connection V3
portfolio Current portfolio data for Bloomberg connection V3
realtime Real-time data for Bloomberg connection V3
stop Unsubscribe real-time requests for Bloomberg connection V3
tahistory Historical technical analysis for Bloomberg connection V3
timeseries Intraday tick data for Bloomberg connection V3

## Bloomberg Server Data Information

category Field category search for Bloomberg connection V3
fieldinfo Field information for Bloomberg connection V3

| | |
|---|---|
| fieldsearch | Field search for Bloomberg connection V3 |
| lookup | Find information about securities for Bloomberg connection V3 |

## Examples

### Connect to Bloomberg Server

Connect to the Bloomberg Server using the IP address of the machine running the Bloomberg Server. This example assumes the following:

- The Bloomberg UUID is `12345678`.
- The IP address for the machine running the Bloomberg Server is `'111.11.11.111'`.

```
uuid = 12345678;
ipaddress = '111.11.11.111';

c = blpsrv(uuid,ipaddress)
```

```
c =

  blpsrv with properties:

               Uuid: 12345678
               User: [1x1 com.bloomberglp.blpapi.impl.aT]
             Userip: '111.11.11.112'
            Session: [1x1 com.bloomberglp.blpapi.Session]
          IPAddress: '111.11.11.111'
               Port: 8194
            TimeOut: 0
       DatetimeType: ''
    DataReturnFormat: ''
```

`blpsrv` connects to the machine running the Bloomberg Server using the default port number `8194`. `blpsrv` creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID
- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg Server
- Port number of the machine running the Bloomberg Server
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)
```

```
d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg Server connection.

```
close(c)
```

### Connect to Bloomberg Server with Port Number

Connect to the Bloomberg Server using the IP address of the machine running the Bloomberg Server. This example assumes the following:

- The Bloomberg UUID is `12345678`.
- The IP address for the machine running the Bloomberg Server is `'111.11.11.111'`.
- The port number of the machine running the Bloomberg Server is `8194`.

```
uuid = 12345678;
ipaddress = '111.11.11.111';
port = 8194;

c = blpsrv(uuid,ipaddress,port)

c =

  blpsrv with properties:

               Uuid: 12345678
               User: [1x1 com.bloomberglp.blpapi.impl.aT]
             Userip: '111.11.11.112'
            Session: [1x1 com.bloomberglp.blpapi.Session]
          IPAddress: '111.11.11.111'
               Port: 8194
            TimeOut: 0
        DatetimeType: ''
    DataReturnFormat: ''
```

`blpsrv` connects to the machine running the Bloomberg Server using the port number `8194` and creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID
- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg Server
- Port number of the machine running the Bloomberg Server

- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)

d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg Server connection.

```
close(c)
```

### Connect to Bloomberg Server with Timeout

Connect to the Bloomberg Server using the IP address of the machine running the Bloomberg Server. This example assumes the following:

- The Bloomberg UUID is `12345678`.
- The IP address for the machine running the Bloomberg Server is `'111.11.11.111'`.
- The port number of the machine running the Bloomberg Server is your default port number.
- The timeout value is 10 milliseconds.

```
uuid = 12345678;
ipaddress = '111.11.11.111';
port = [];
timeout = 10;

c = blpsrv(uuid,ipaddress,port,timeout)

c =

  blpsrv with properties:

              Uuid: 12345678
              User: [1x1 com.bloomberglp.blpapi.impl.aT]
            Userip: '111.11.11.112'
           Session: [1x1 com.bloomberglp.blpapi.Session]
         IPAddress: '111.11.11.111'
              Port: 8194
           TimeOut: 10
```

```
        DatetimeType: ''
    DataReturnFormat: ''
```

`blpsrv` connects to the machine running the Bloomberg Server using the default port number `8194` and a timeout value of 10 milliseconds. `blpsrv` creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID
- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg Server
- Port number of the machine running the Bloomberg Server
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)

d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg Server connection.

```
close(c)
```

## See Also

**Topics**

“Connect to Bloomberg” on page 3-2
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-3
“Workflow for Bloomberg” on page 3-19

**Introduced in R2014b**

# bpipe

Bloomberg B-PIPE connection V3

## Description

The `bpipe` function creates a `bpipe` object. The `bpipe` object represents a Bloomberg B-PIPE connection.

Other functions connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg Server (`blpsrv`), and Bloomberg Data License (`bdl`). For details about these services, see “Comparing Bloomberg Connections” on page 2-3.

For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing `bpipe`. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

## Creation

### Syntax

```
c = bpipe(authtype,appname,ipaddress,port)
c = bpipe(authtype,appname,ipaddress,port,timeout)
c = bpipe(authtype,appname,ipaddress,port,timeout,tlscred,tlspassword,
tlstrust)
```

#### Description

`c = bpipe(authtype,appname,ipaddress,port)` creates a Bloomberg B-PIPE connection object `c`, and sets these properties:

- authtype
- appname
- ipaddress
- port

`c = bpipe(authtype,appname,ipaddress,port,timeout)` also sets the timeout property.

`c = bpipe(authtype,appname,ipaddress,port,timeout,tlscred,tlspassword, tlstrust)` connects to a B-PIPE zero-footprint cloud solution using the specified credentials file, password, and trust file.

---

**Caution:** To refer to a Bloomberg connection in other functions, use the connection object created by the `bpipe` function. Otherwise, using `bpipe` as an input argument opens multiple Bloomberg connections, causing unexpected behavior and exhausting memory resources.

---

### Input Arguments

#### `tlscred` — Credentials file
character vector | string scalar

Credentials file, specified as a character vector or string scalar that contains the full path to the credentials file with the extension `pk12`. For details about the credentials file, contact Bloomberg.

Data Types: `char` | `string`

#### `tlspassword` — B-PIPE password
character vector | string scalar

B-PIPE password, specified as a character vector or string scalar. To obtain your B-PIPE password, contact Bloomberg.

Data Types: `char` | `string`

#### `tlstrust` — Trust file
character vector | string scalar

Trust file, specified as a character vector or string scalar that contains the full path to the trust file with the extension `pk7`. For details about the trust file, contact Bloomberg.

Data Types: `char` | `string`

## Properties

#### `AppAuthType` — Application authentication type
`""` (default) | `"APPNAME_AND_KEY"`

This property is read-only.

Application authentication type, specified as one of these values:

- `""` — Bloomberg B-PIPE connection with Windows authentication
- `"APPNAME_AND_KEY"` — Bloomberg B-PIPE connection with application authentication

#### `AuthType` — Bloomberg user authentication type
`"OS_LOGON"` | `"APPLICATION_ONLY"`

Bloomberg user authentication type, specified as one of these values:

- `"OS_LOGON"` — Bloomberg B-PIPE connection with Windows authentication
- `"APPLICATION_ONLY"` — Bloomberg B-PIPE connection with application authentication

For details, see the *Bloomberg B-PIPE API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

#### `AppName` — Application name
character vector | string

Application name, specified as a character vector or string that identifies the application you are using to connect to Bloomberg B-PIPE.

Example: `'appname'`

Data Types: `char` | `string`

**`User` — Bloomberg user**
Bloomberg user identity object

This property is read-only.

Bloomberg user, specified as a Bloomberg user identity object.

Example: `[1x1 com.bloomberglp.blpapi.impl.aT]`

**`Session` — Bloomberg V3 session**
Bloomberg V3 API Session object

This property is read-only.

Bloomberg V3 session, specified as a Bloomberg V3 API Session object.

Example: `[1x1 com.bloomberglp.blpapi.Session]`

**`IPAddress` — IP address**
character vector | cell array of character vectors | string | string array

IP address of the machine running the Bloomberg B-PIPE process, specified as a character vector, cell array of character vectors, string, or string array. A character vector or string identifies the machine running the Bloomberg B-PIPE process, whereas a cell array of character vectors or string array specifies multiple machines.

Example: `{'111.11.11.112'}`

Data Types: `char` | `cell` | `string`

**`Port` — Port number**
`[]` (default) | numeric scalar

Port number of the machine running the Bloomberg B-PIPE process, specified as a numeric scalar.

Example: `8194`

Data Types: `double`

**`TimeOut` — Timeout**
numeric scalar

Timeout specifying the time in milliseconds that MATLAB attempts to connect to the machine running the Bloomberg B-PIPE process before timing out, specified as a numeric scalar.

Example: `1000`

Data Types: `double`

**`DatetimeType` — Date and time data type**
`''` (default) | `'datetime'`

Date and time data type, specified as one of these values.

| Value | Description |
|---|---|
| '' (default) | Return date and time values as MATLAB date numbers. |
| 'datetime' | Return date and time values as a datetime array. |

You can specify these values using a character vector or string (for example, "datetime").

When you create a bpipe object, the bpipe function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DatetimeType = 'datetime';
```

Then, you can use these supported functions:

- getbulkdata
- getdata
- history
- tahistory
- timeseries

> **Note** If the DataReturnFormat property value is 'table' and the DatetimeType property value is 'datetime', then the returned data is a table that contains date and time values as a datetime array. If the DataReturnFormat property value is an empty character vector, then setting the DatetimeType property to 'datetime' returns date and time values for aggregated ticks and historical requests as MATLAB date numbers.

**DataReturnFormat — Data return format**
'cell' | 'structure' | 'table' | 'timetable'

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| 'cell' | cell array |
| 'table' | table |
| 'timetable' | timetable |
| 'structure' | structure |

**Note** The default data type of the returned data depends on the executed function. To specify the default data type, set the `DataReturnFormat` property to `''`. For default data types, see the supported function list.

You can specify these values using a character vector or string (for example, `"table"`).

When you create a `bpipe` object, the `bpipe` function leaves this property unset. To retrieve data, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'structure';
```

Then, you can use these supported functions.

| Supported Function | Valid Data Types for Returned Data |
|---|---|
| `category` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `eqs` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `fieldinfo` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `fieldsearch` | • `cell array` (default)<br>• `structure`<br>• `table` |
| `lookup` | • `structure` (default)<br>• `table` |
| `portfolio` | • `structure` (default)<br>• `table` |
| `getbulkdata` | • `structure` (default)<br>• `table`<br>• `timetable` |
| `getdata` | • `structure` (default)<br>• `table`<br>• `timetable` |
| `history` | • numeric array (default)<br>• `table`<br>• `timetable` |

| Supported Function | Valid Data Types for Returned Data |
|---|---|
| `tahistory` | • `structure` (default)<br>• `table`<br>• `timetable` |
| `timeseries` | • `cell array` (default for raw tick data)<br>• numeric array (default for interval tick data)<br>• `table`<br>• `timetable` |

**Note** Regardless of the `DatetimeType` property value, if the `DataReturnFormat` property value is `'timetable'`, then the `getdata` and `getbulkdata` functions return a table that contains date and time values as `datetime` arrays.

## Object Functions

## Bloomberg B-PIPE Connection

| | |
|---|---|
| close | Close Bloomberg connection V3 |
| get | Properties of Bloomberg connection V3 |
| isconnection | Determine Bloomberg connection V3 |

## Bloomberg B-PIPE Data Retrieval

| | |
|---|---|
| eqs | Equity screening data for Bloomberg connection V3 |
| getbulkdata | Bulk data with header information for Bloomberg connection V3 |
| getdata | Current data for Bloomberg connection V3 |
| history | Historical data for Bloomberg connection V3 |
| portfolio | Current portfolio data for Bloomberg connection V3 |
| realtime | Real-time data for Bloomberg connection V3 |
| stop | Unsubscribe real-time requests for Bloomberg connection V3 |
| tahistory | Historical technical analysis for Bloomberg connection V3 |
| timeseries | Intraday tick data for Bloomberg connection V3 |

## Bloomberg B-PIPE Data Information

| | |
|---|---|
| category | Field category search for Bloomberg connection V3 |
| fieldinfo | Field information for Bloomberg connection V3 |
| fieldsearch | Field search for Bloomberg connection V3 |
| lookup | Find information about securities for Bloomberg connection V3 |

## Examples

### Create Bloomberg B-PIPE Connection

Create a Bloomberg B-PIPE connection using the IP address of the machine running the Bloomberg B-PIPE process. This example assumes the following:

- The authentication is Windows authentication when you set `authtype` to `'OS_LOGON'`.
- The application name is blank because you are not connecting to Bloomberg B-PIPE using an application.
- The IP address for the machine running the Bloomberg B-PIPE process is `'111.11.11.112'`.
- The port number of the machine running the Bloomberg B-PIPE process is `8194`.

```
authtype = 'OS_LOGON';
appname = '';
ipaddress = {'111.11.11.112'};
port = 8194;

c = bpipe(authtype,appname,ipaddress,port)
```

```
c =

  bpipe with properties:

         AppAuthType: ''
            AuthType: 'OS_LOGON'
             AppName: []
                User: [1x1 com.bloomberglp.blpapi.impl.aT]
             Session: [1x1 com.bloomberglp.blpapi.Session]
           IPAddress: {'111.11.11.112'}
                Port: 8194.00
             TimeOut: 0
        DatetimeType: ''
    DataReturnFormat: ''
```

`bpipe` connects to the machine running Bloomberg B-PIPE at port number `8194`. `bpipe` creates the Bloomberg B-PIPE connection object `c` with these properties:

- Application authentication type
- Bloomberg user authentication type
- Application name
- Bloomberg user identity object
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg B-PIPE process
- Port number of the machine running the Bloomberg B-PIPE process
- Number (in milliseconds) specifying how long MATLAB attempts to connect to the machine before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
```

```
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)

d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg B-PIPE connection.

```
close(c)
```

### Create Bloomberg B-PIPE Connection with Timeout

Create a Bloomberg B-PIPE connection using the IP address of the machine running the Bloomberg B-PIPE process. This example assumes the following:

- The authentication is Windows authentication when you set `authtype` to `'OS_LOGON'`.
- The application name is blank because you are not connecting to Bloomberg B-PIPE using an application.
- The IP address for the machine running the Bloomberg B-PIPE process is `'111.11.11.112'`.
- The port number of the machine running the Bloomberg B-PIPE process is `8194`.
- The timeout value is 1000 milliseconds.

```
authtype = 'OS_LOGON';
appname = '';
ipaddress = {'111.11.11.112'};
port = 8194;
timeout = 1000;

c = bpipe(authtype,appname,ipaddress,port,timeout)

c =

  bpipe with properties:

         AppAuthType: ''
            AuthType: 'OS_LOGON'
             AppName: []
                User: [1x1 com.bloomberglp.blpapi.impl.aT]
             Session: [1x1 com.bloomberglp.blpapi.Session]
           IPAddress: {'172.28.17.118'}
                Port: 8194.00
             TimeOut: 1000.00
        DatetimeType: ''
    DataReturnFormat: ''
```

`bpipe` connects to the machine running Bloomberg B-PIPE at port number `8194`. `bpipe` creates the Bloomberg B-PIPE connection object `c` with these properties:

- Application authentication type
- Bloomberg user authentication type
- Application name
- Bloomberg user identity object
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg B-PIPE process
- Port number of the machine running the Bloomberg B-PIPE process
- Number (in milliseconds) specifying how long MATLAB attempts to connect to the machine before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)
```

```
d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg B-PIPE connection.

```
close(c)
```

### Create Bloomberg B-PIPE Zero-Footprint Connection

Create a Bloomberg B-PIPE zero-footprint connection using the IP address of the machine running the Bloomberg B-PIPE process. This example assumes the following:

- The authentication is based on the application name when you set `authtype` to `'APPLICATION_ONLY'`.
- The application name is `'APP'`.
- The IP address for the machine running the Bloomberg B-PIPE process is `'111.11.11.112'`.
- The port number of the machine running the Bloomberg B-PIPE process is `8194`.
- The number (in milliseconds) specifying how long MATLAB attempts to connect to the machine before timing out is 1000.
- The full path of the credentials file is `C:\ABCDEFG.pk12`.
- The B-PIPE password is `12345`.
- The full path of the trust file is `C:\HIJKLM.pk7`.

```
authtype = 'APPLICATION_ONLY';
appname = 'APP';
ipaddress = {'111.11.11.112'};
port = 8194;
timeout = 1000;
tlscred = 'C:\ABCDEFG.pk12';
tlspassword = '12345';
tlstrust = 'C:\HIJKLM.pk7';

c = bpipe(authtype,appname,ipaddress,port,timeout,tlscred,tlspassword,tlstrust)

c =

  bpipe with properties:

         AppAuthType: 'APPNAME_AND_KEY'
            AuthType: 'APPLICATION_ONLY'
             AppName: 'APP'
                User: [1x1 com.bloomberglp.blpapi.impl.by]
             Session: [1x1 com.bloomberglp.blpapi.Session]
           IPAddress: {'111.11.11.112'}
                Port: 8194.00
             TimeOut: 1000.00
        DatetimeType: ''
    DataReturnFormat: ''
```

`bpipe` connects to the machine running Bloomberg B-PIPE at port number `8194`. The `bpipe` function creates the Bloomberg B-PIPE connection object `c` with these properties:

- Application authentication type
- Bloomberg user authentication type
- Application name
- Bloomberg user identity object
- Bloomberg V3 API Session object
- IP address of the machine running the Bloomberg B-PIPE process
- Port number of the machine running the Bloomberg B-PIPE process
- Number (in milliseconds) specifying how long MATLAB attempts to connect to the machine before timing out
- Date and time data type
- Data return format

Request the last and open prices for Microsoft.

```
format bank  % Display data format for currency
s = 'MSFT US Equity';
f = {'LAST_PRICE';'OPEN'};
[d,sec] = getdata(c,s,f)

d =
    LAST_PRICE: 33.34
          OPEN: 33.60

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the Bloomberg B-PIPE connection.

```
close(c)
```

## See Also

**Topics**

“Connect to Bloomberg” on page 3-2
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-3
“Workflow for Bloomberg” on page 3-19

**Introduced in R2014b**

# bdl

Bloomberg Data License connection

## Description

The `bdl` function creates a `bdl` object. The `bdl` object represents a Bloomberg Data License connection.

Other functions connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg Server (`blpsrv`), and Bloomberg B-PIPE (`bpipe`). For details about these services, see “Comparing Bloomberg Connections” on page 2-3.

For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3. To ensure a successful Bloomberg connection, perform the required steps before executing `bdl`. For details, see “Installing Bloomberg and Configuring Connections” on page 1-5.

## Creation

### Syntax

```
c = bdl(login,password,hostname,port,decrypt)
c = bdl(login,password,hostname,port,decrypt,authoption,keyfile,passphrase)
```

#### Description

`c = bdl(login,password,hostname,port,decrypt)` connects to the Bloomberg Data License server using the Secure File Transfer Protocol (SFTP). This syntax sets the Login, Hostname, and Port properties, and uses these input arguments:

- Bloomberg Data License SFTP server password
- Decryption code `decrypt`

`c = bdl(login,password,hostname,port,decrypt,authoption,keyfile,passphrase)` connects to the Bloomberg Data License server using key authentication, sets the AuthOption and KeyFile properties, and uses a pass phrase.

#### Input Arguments

**`password` — Bloomberg Data License SFTP password**
character vector | string scalar

Bloomberg Data License SFTP password, specified as a character vector or string scalar.

Example: `'xxxxxxxx'`

Data Types: `char` | `string`

**`decrypt` — Decryption code**
character vector | string scalar

Decryption code, specified as a character vector or string scalar to denote the Data Encryption Standard (DES) encryption key.

Example: `'nAcLeZ'`

Data Types: `char` | `string`

**`passphrase` — Pass phrase**
character vector | string scalar

Pass phrase, specified as a character vector or string scalar. The `bdl` function uses this phrase to decrypt the key file. Use this argument only when the authentication type is set to `'key'`.

Example: `'mykeyphrase'`

Data Types: `char` | `string`

## Properties

**`Login` — Bloomberg Data License SFTP server login name**
`''` (default) | character vector | string scalar

Bloomberg Data License SFTP server login name, specified as a character vector or string scalar.

Example: `'xxxxx'`

Data Types: `char` | `string`

**`Hostname` — Bloomberg Data License SFTP server name**
`''` (default) | character vector | string scalar

Bloomberg Data License SFTP server name, specified as a character vector or string scalar.

Example: `'dlsftp.bloomberg.com'`

Data Types: `char` | `string`

**`Port` — Bloomberg Data License SFTP port number**
numeric scalar

Bloomberg Data License SFTP port number of the machine running the Bloomberg Data License server, specified as a numeric scalar.

Example: `30206`

Data Types: `double`

**`AuthOption` — Authentication type**
`'password'` (default) | `'key'`

Authentication type, specified as one of these values.

- `'password'` — Password authentication
- `'key'` — Key file authentication

When you use the `'password'` value, you must specify the Bloomberg Data License SFTP server password in the `password` input argument.

When you use the `'key'` value, you must specify a key file name by setting the `KeyFile` property and specify a pass phrase in the `passphrase` input argument.

**`KeyFile` — Key file**
`''` (default) | character vector | string scalar

Key file, specified as a character vector or string scalar to denote the full path for the private key file. Use this property only when the authentication type is set to `'key'`.

Example: `'c:\temp\mykeyfile'`

Data Types: `char` | `string`

**`Connection` — Bloomberg Data License API connection**
Bloomberg Data License API connection object

Bloomberg Data License API connection, specified as a Bloomberg Data License API connection object.

Example: `[1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]`

## Object Functions

close Close Bloomberg connection V3
dir Current Bloomberg Data License connection folder listing

## Examples

### Connect to Bloomberg Data License Using Password Authentication

Connect to the Bloomberg Data License SFTP server using a password.

Create the Bloomberg Data License connection `c`. This example assumes the following:

- The Bloomberg Data License SFTP server login name is `'xxxxx'`.
- The Bloomberg Data License SFTP server password is `'xxxxxxxx'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.

```
login = 'xxxxx';
password = 'xxxxxxxx';
hostname = 'dlsftp.bloomberg.com';
port = 30206;
decrypt = 'nAcLeZ';

c = bdl(login,password,hostname,port,decrypt)
```

```
c =

bdl with properties:

          Login: 'xxxxx'
       Hostname: 'dlsftp.bloomberg.com'
```

```
          Port: 30206
    AuthOption: 'password'
       KeyFile: ''
    Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

`c` returns the Bloomberg Data License connection object with these properties:

- Bloomberg Data License SFTP server login name
- Bloomberg Data License SFTP server name
- Bloomberg Data License SFTP port number
- Authentication type (in this case, the default password authentication)
- Key file (in this case, blank)
- Bloomberg Data License API object

For an example of data retrieval, see “Retrieve Data Using Bloomberg Data License” on page 3-16.

Close the Bloomberg Data License connection.

```
close(c)
```

### Connect to Bloomberg Data License Using Key File

Connect to the Bloomberg Data License SFTP server using a key file.

Create the Bloomberg Data License connection `c`. This example assumes the following:

- The Bloomberg Data License SFTP server login name is `'xxxxx'`.
- The Bloomberg Data License SFTP server password is `'xxxxxxxx'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.
- The authentication type is `'key'`.
- The full path to the key file is `'c:\temp\mykeyfile'`.
- The pass phrase is `'mykeyphrase'`.

```
login = 'xxxxx';
password = 'xxxxxxxx';
hostname = 'dlsftp.bloomberg.com';
port = 30206;
decrypt = 'nAcLeZ';
authoption = 'key';
keyfile = 'c:\temp\mykeyfile';
passphrase = 'mykeyphrase';

c = bdl(login,password,hostname,port,decrypt,authoption, ...
    keyfile,passphrase)

c =

bdl with properties:
```

```
          Login: 'xxxxx'
       Hostname: 'dlsftp.bloomberg.com'
           Port: 30206
     AuthOption: 'key'
        KeyFile: 'c:\temp\mykeyfile'
     Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

`c` returns the Bloomberg Data License connection object with these properties:

- Bloomberg Data License SFTP server login name
- Bloomberg Data License SFTP server name
- Bloomberg Data License SFTP port number
- Authentication type (in this case, key authentication)
- Full path to the key file
- Bloomberg Data License API object

For an example of data retrieval, see “Retrieve Data Using Bloomberg Data License” on page 3-16.

Close the Bloomberg Data License connection.

```
close(c)
```

## Tips

For details about Bloomberg Data License, see the relevant guides by entering `DLSD` and clicking **<GO>** in the Bloomberg terminal.

## See Also

**Topics**

“Connect to Bloomberg” on page 3-2
“Retrieve Data Using Bloomberg Data License” on page 3-16
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-3
“Workflow for Bloomberg” on page 3-19

**Introduced in R2015a**

# dir

Current Bloomberg Data License connection folder listing

## Syntax

```
list = dir(c)
```

## Description

`list = dir(c)` returns the contents of the current working folder using the Bloomberg Data License connection `c`.

## Examples

### Request the Bloomberg Data License Folder Listing

Create the Bloomberg Data License connection `c`. This code assumes the following:

- The Bloomberg Data License SFTP server login name is `'xxxxx'`.
- The Bloomberg Data License SFTP server password is `'xxxxxxxx'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.

```
username = 'xxxxx';
password = 'xxxxxxxx';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'nAcLeZ';

c = bdl(username,password,hostname,portnumber,decrypt);
```

Request the contents `list` of the current working folder using `c`.

```
list = dir(c)
```

```
list =

  'd--x--x--x    2 root     root         4096 Sep  5 11:25 bin'
  'dr--r--r--    2 root     root         4096 Sep  5 11:25 etc'
  'drwxrwxrwx    2 op       general     61440 Sep 24 02:11 notices'
  ...
```

`list` contains a cell array of character vectors. Each character vector is one folder or file name.

Close the Bloomberg Data License connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg Data License connection

connection object

Bloomberg Data License connection, specified as a connection object created using `bdl`.

## Output Arguments

### `list` — Current working folder contents

cell array of character vectors

Current working folder contents, returned as an `n`-by-1 cell array of character vectors. `n` is the number of files or folders within the current working folder. Each character vector contains folder listing information for a folder or file name.

## Tips

For details about Bloomberg Data License, see the relevant guides by entering `DLSD` and clicking **<GO>** in the Bloomberg terminal.

## See Also

`bdl` | `close`

**Topics**

“Retrieve Data Using Bloomberg Data License” on page 3-16
“Workflow for Bloomberg” on page 3-19

**Introduced in R2015a**

# category

Field category search for Bloomberg connection V3

## Syntax

```
d = category(c,f)
```

## Description

`d = category(c,f)` returns category information given the search term `f`.

## Examples

### Search for Bloomberg Last Price Field

Create a Bloomberg® connection, and then request the category description of the last price field.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `category` function returns data as a cell array.

```
c.DataReturnFormat = 'table';
```

Request the Bloomberg category description of the last price field.

```
f = 'LAST_PRICE';
d = category(c,f);
```

Display the first three rows of the Bloomberg category description data in `d`.

```
d(1:3,:)
```

```
ans =

  3×5 table

    CATEGORY       ID                 MNEMONIC                          DESCRIPTION
    __________    _______    ____________________________    ____________________________

    'Analysis'    'OP179'    'THETA_LAST'                    'Theta Last Price'
    'Analysis'    'VM048'    'DDMX_PERCENT_CHANGE_LAST_PRICE'    'DDMX Percent Change Last Price
    'Analysis'    'YL005'    'YLD_CNV_LAST'                  'Last Yield To Convention'
```

The columns in `d` are:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### f — Search term

character vector | string scalar

Search term, specified as a character vector or string scalar to denote Bloomberg fields.

Data Types: `char` | `string`

## Output Arguments

### d — Category data

cell array (default) | structure | table

Category data, returned as an N-by-5 cell array, a structure, or a table.

The columns (or fields) of the data types are:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

The data type of the category data depends on the DataReturnFormat property of the connection object.

## See Also

`blp` | `close` | `fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

**Topics**

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010b**

# close

Close Bloomberg connection V3

## Syntax

```
close(c)
```

## Description

`close(c)` closes the Bloomberg connection V3 `c`.

## Examples

### Close the Bloomberg Connection

Create the Bloomberg connection object `c` using `blp`.

```
c = blp;
```

Alternatively, you can establish these connections:

- Bloomberg Server using `blpsrv`
- Bloomberg B-PIPE using `bpipe`
- Bloomberg Data License using `bdl`

Close the Bloomberg connection using the Bloomberg connection object `c`.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as one of these connection objects:

- Bloomberg connection V3 created using `blp`
- Bloomberg Server connection created using `blpsrv`
- Bloomberg B-PIPE connection created using `bpipe`
- Bloomberg Data License connection created using `bdl`

## See Also

`bdl` | `blp` | `blpsrv` | `bpipe`

### Topics

“Connect to Bloomberg” on page 3-2
“Retrieve Bloomberg Current Data” on page 3-6

“Retrieve Bloomberg Historical Data” on page 3-8
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Retrieve Bloomberg Intraday Tick Data” on page 3-12
“Retrieve Bloomberg Real-Time Data” on page 3-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010a**

# eqs

Equity screening data for Bloomberg connection V3

## Syntax

```
d = eqs(c,sname)
d = eqs(c,sname,stype)
d = eqs(c,sname,stype,languageid)
d = eqs(c,sname,stype,languageid,group)
d = eqs(c,sname,stype,languageid,group,'OverrideFields',ov)
```

## Description

`d = eqs(c,sname)` returns equity screening data given the Bloomberg V3 session screen name `sname`.

`d = eqs(c,sname,stype)` also specifies the screen type `stype`.

`d = eqs(c,sname,stype,languageid)` also specifies the language identifier `languageid`.

`d = eqs(c,sname,stype,languageid,group)` also specifies the optional group identifier `group`.

`d = eqs(c,sname,stype,languageid,group,'OverrideFields',ov)` also specifies the Bloomberg override fields and values `ov`.

## Examples

### Retrieve Equity Screening Data for Screen

Create a Bloomberg® connection, and then retrieve frontier market stock data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `eqs` function returns data as a cell array.

```
c.DataReturnFormat = 'table';
```

Retrieve equity screening data for the screen named `Frontier Market Stocks with 1 billion USD Market Caps`.

```
sname = 'Frontier Market Stocks with 1 billion USD Market Caps';
d = eqs(c,sname);
```

Display the first three rows in the returned data `d`.

```
d(1:3,:)
```

```
ans =

  3×8 table

      Cntry                 Name                  IndGroup    MarketCap     Price_D_1     P_B
    __________    ___________________________    ________    __________    _________    _____

    'Venezuela'    'MERCANTIL SERVICIOS FINAN-A'    'Banks'     7.3424e+12      70088      278.25
    'Venezuela'    'BANCO DEL CARIBE-A'             'Banks'     2.0442e+12      24531      2321.8
    'Venezuela'    'BANCO PROVINCIAL'               'Banks'     1.2632e+12      11715      52.344
```

The columns in d are:

- Country name
- Company name
- Industry name
- Market capitalization
- Price
- Price-to-book ratio
- Price-earnings ratio
- Earnings per share

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Equity Screening Data for a Screen Type

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve equity screening data for the screen called `Vehicle-Engine-Parts` and the screen type equal to `'GLOBAL'`.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL');
```

Display the first three rows in the returned data `d`.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'        'Price:D-1'    'P/E'
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36]    [17.28]
```

```
  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'      'EPS T12M'
    [          42.43]    [38248998912.00]    [    4.11]
    [          24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

### Retrieve Equity Screening Data for a Screen in German

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve equity screening data for the screen called `Vehicle-Engine-Parts`, the screen type equal to `'GLOBAL'`, and return data in German.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','GERMAN');
```

Display the first three rows in the returned data `d`.

```
d(1:3,:)
```

```
  Columns 1 through 5

    'Ticker'        'Kurzname'          'Marktkapitalisie...'    'Preis:D-1'    'KGV'
    'HON     US'    'HONEYWELL INTL'    [     69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [     24799526912.00]    [   132.36]    [17.28]

  Columns 6 through 8

    'Gesamtertrag YTD'    'Erlös T12M'        'EPS T12M'
    [          42.43]    [38248998912.00]    [    4.11]
    [          24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers in German. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

### Retrieve Equity Screening Data for a Screen with a Specified Screen Folder Name

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve equity screening data for the Bloomberg screen called `Vehicle-Engine-Parts`, using the Bloomberg screen type `'GLOBAL'` and the language `'ENGLISH'`, and the Bloomberg screen folder name `'GENERAL'`.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','ENGLISH','GENERAL');
```

Display the first three rows in the returned data `d`.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'        'Price:D-1'    'P/E'
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36]    [17.28]

  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'      'EPS T12M'
    [           42.43]    [38248998912.00]    [    4.11]
    [           24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio

- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

### Retrieve Equity Screening Data Using Override Fields

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve equity screening data as of a specified date using these input arguments. The override field `PiTDate` is equivalent to the flag `AsOf` in the Bloomberg Excel Add-In.

- Bloomberg connection `c`
- Bloomberg screen is `Vehicle-Engine-Parts`
- Bloomberg screen type is `'GLOBAL'`
- Language is `'ENGLISH'`
- Bloomberg screen folder name is `'GENERAL'`
- Override field `PiTDate` is September 9, 2014

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','ENGLISH','GENERAL',...
    'OverrideFields',{'PiTDate','20140909'});
```

Display the first three rows in the returned data `d`.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'    'Price:D-1'       'P/E'
    'HON US'     'HONEYWELL INTL'      [7.3919e+10]    [  94.4600]    [17.8087]
    'TSLA US'    'TESLA MOTORS'        [3.4707e+10]    [ 278.4800]    [    NaN]

  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'    'EPS T12M'
    [          4.8907]    [  3.9966e+10]    [  5.1600]
    [         85.1239]    [  2.4365e+09]    [ -1.3500]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen as of September 9, 2014. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization

- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `sname` — Screen name

character vector | string scalar

Screen name, specified as a character vector or string scalar to denote the Bloomberg V3 session screen name to execute. The screen can be a customized equity screen or one of the Bloomberg example screens accessed by using the **EQS <GO>** option from the Bloomberg terminal.

Data Types: `char` | `string`

### `stype` — Screen type

`'GLOBAL'` | `'PRIVATE'`

Screen type, specified as one of the two preceding values to denote the Bloomberg screen type. `'GLOBAL'` denotes a Bloomberg screen name and `'PRIVATE'` denotes a customized screen name. When using the optional `group` input argument, `stype` cannot be set to `'PRIVATE'` for customized screen names.

### `languageid` — Language identifier

character vector | string scalar

Language identifier, specified as a character vector or string to denote the language for the returned data. This argument is optional.

Data Types: `char` | `string`

### `group` — Group identifier

character vector | string scalar

Group identifier, specified as a character vector or string to denote the Bloomberg screen folder name accessed by using the **EQS <GO>** option from the Bloomberg terminal. This argument is optional. When using this argument, `stype` cannot be set to `'PRIVATE'` for customized screen names.

Data Types: `char` | `string`

### `ov` — Bloomberg override field values

cell array

Bloomberg override field values, specified as an n-by-2 cell array. The first column of the cell array is the override field. The second column is the override value.

Example: `{'PiTDate','20140909'}`

Data Types: `cell`

## Output Arguments

### d — Equity screening data

cell array (default) | structure | table

Equity screening data, returned as a cell array, structure, or table. The data type of the equity screening data depends on the DataReturnFormat property of the connection object.

## See Also

`blp` | `close` | `getdata` | `tahistory`

**Topics**

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2012b**

# fieldinfo

Field information for Bloomberg connection V3

## Syntax

```
d = fieldinfo(c,f)
```

## Description

`d = fieldinfo(c,f)` returns field information on the Bloomberg V3 connection object `c` given the field mnemonic `f`.

## Examples

### Retrieve Information for Last Price Field

Create a Bloomberg® connection, and then retrieve information for the last price field.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `fieldinfo` function returns data as a cell array.

```
c.DataReturnFormat = 'table';
```

Retrieve the Bloomberg field information for the `LAST_PRICE` field.

```
f = 'LAST_PRICE';
d = fieldinfo(c,f);
```

Display the last four columns in the returned Bloomberg information.

```
d(:,2:5)
```

```
ans =

  1×4 table

     ID         MNEMONIC            DESCRIPTION          DATATYPE
    _______    ____________    ________________________    ________

    'RQ005'    'LAST_PRICE'    'Last Trade/Last Price'    'Double'
```

The columns in `d` are:

- Field identifier
- Field mnemonic
- Field name
- Field data type

You can also access the Bloomberg help information in the first column.

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### f — Field mnemonic

character vector | string scalar

Field mnemonic, specified as a character vector or string scalar that denotes the Bloomberg field information to retrieve.

Data Types: `char` | `string`

## Output Arguments

### d — Field information

cell array (default) | structure | table

Field information, returned as an N-by-5 cell array, a structure, or a table.

The columns (or fields) of the data types are:

- Field help
- Field identifier
- Field mnemonic
- Field name
- Field data type

The data type of the field information depends on the DataReturnFormat property of the connection object.

## See Also

`blp` | `category` | `close` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

### Topics

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010b**

# fieldsearch

Field search for Bloomberg connection V3

## Syntax

```
d = fieldsearch(c,f)
```

## Description

`d = fieldsearch(c,f)` returns field information on the Bloomberg V3 connection object `c` given the search term `f`.

## Examples

### Search for Last Price Field Information

Create a Bloomberg® connection, and then return information for the last price field.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `fieldsearch` function returns data as a cell array.

```
c.DataReturnFormat = 'table';
```

Return information for the search term `LAST_PRICE`.

```
f = 'LAST_PRICE';
d = fieldsearch(c,f);
```

Display the first three rows of the field information in `d`.

```
d(1:3,:)
```

```
ans =

  3×5 table

          CATEGORY             ID            MNEMONIC                DESCRIPTION
    ___________________________    _______    ____________________    ___________________________________

    'Market Activity/Last'       'PR005'    'PX_LAST'               'Last Price'
    'Market Activity/Last'       'RQ005'    'LAST_PRICE'            'Last Trade/Last Price'
    'Market Activity/Last'       'PR910'    'CRNCY_ADJ_PX_LAST'     'Currency Adjusted Last Price'
```

The columns in `d` are:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `f` — Search term

character vector | string scalar

Search term, specified as a character vector or string scalar that denotes the Bloomberg field descriptive data to retrieve.

Data Types: `char` | `string`

## Output Arguments

### `d` — Field data

cell array (default) | structure | table

Field data, returned as an N-by-5 cell array, a structure, or a table.

The columns (or fields) of the data types are:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

The data type of the field data depends on the DataReturnFormat property of the connection object.

## See Also

`blp` | `category` | `close` | `fieldinfo` | `getdata` | `history` | `realtime` | `timeseries`

**Topics**

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010b**

# get

Properties of Bloomberg connection V3

## Syntax

```
v = get(c)
v = get(c,properties)
```

## Description

`v = get(c)` returns a structure where each field name is the name of a property of `c` and each field contains the value of that property.

`v = get(c,properties)` returns the value of the specified properties `properties` for the Bloomberg V3 connection object.

## Examples

### Retrieve Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the Bloomberg connection properties.

```
v = get(c)
```

```
v =

      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: 'localhost'
         port: 8194
      timeout: 0
```

`v` is a structure containing the Bloomberg session object, IP address, port number, and timeout value.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve One Bloomberg Connection Property

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the port number from the Bloomberg connection object by specifying `'port'` as a character vector.

```
property = 'port';
v = get(c,property)
```

```
v =

        8194
```

`v` is a double that contains the port number of the Bloomberg connection object.

Close the Bloomberg connection.

```
close(c)
```

#### Retrieve Two Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Create a cell array `properties` with character vectors `'session'` and `'port'`. Retrieve the Bloomberg session object and port number from the Bloomberg connection object.

```
properties = {'session','port'};
v = get(c,properties)
```

```
v =

    session: [1x1 com.bloomberglp.blpapi.Session]
       port: 8194
```

`v` is a structure containing the Bloomberg session object and port number.

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `properties` — Property names

character vector | string scalar | cell array of character vectors | string array

Property names, specified as a character vector, string scalar, cell array of character vectors, or string array containing Bloomberg connection property names. The property names are `session`, `ipaddress`, `port`, and `timeout`.

Data Types: `char` | `cell` | `string`

## Output Arguments

### v — Bloomberg connection properties

numeric scalar | character vector | object | structure

Bloomberg connection properties, returned as these data types depending on the requested properties.

| Requested Properties | Data Type |
|---|---|
| Port number or timeout | Numeric scalar |
| IP address | Character vector |
| Bloomberg session | Object |
| All properties | Structure |

## See Also

`blp` | `close` | `getdata` | `history` | `realtime` | `timeseries`

### Topics

“Connect to Bloomberg” on page 3-2
“Workflow for Bloomberg” on page 3-19

### Introduced in R2010a

# getbulkdata

Bulk data with header information for Bloomberg connection V3

## Syntax

```
d = getbulkdata(c,s,f)
d = getbulkdata(c,s,f,o,ov)
d = getbulkdata(c,s,f,o,ov,Name,Value)
[d,sec] = getbulkdata( ___ )
```

## Description

`d = getbulkdata(c,s,f)` returns the bulk data for the fields `f` for the security list `s`.

`d = getbulkdata(c,s,f,o,ov)` returns the bulk data using the override fields `o` with corresponding override values `ov`.

`d = getbulkdata(c,s,f,o,ov,Name,Value)` returns the bulk data with additional options specified by one or more name-value pair arguments for Bloomberg request settings.

`[d,sec] = getbulkdata( ___ )` additionally returns the security list `sec` using any of the input argument combinations in the previous syntaxes.

## Examples

### Return Specific Field for Given Security

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the dividend history for IBM.

```
security = 'IBM US Equity';
field = 'DVD_HIST'; % Dividend history field

[d,sec] = getbulkdata(c,security,field)
```

```
d =

    DVD_HIST: {{149x7 cell}}

sec =

    'IBM US Equity'
```

`d` is a structure with one field that contains a cell array with the returned bulk data. `sec` contains the IBM security name.

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}
```

```
ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       735536]    [ 735544]    [     735546]    [      735578]    [         0.95]      'Quarter'
    [       735445]    [ 735453]    [     735455]    [      735487]    [         0.95]      'Quarter'
    [       735354]    [ 735362]    [     735364]    [      735395]    [         0.95]      'Quarter'
    ...

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    ...
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

### Return Specific Field Using Override Values

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the dividend history for IBM with dividend dates from January 1, 2004 through January 1, 2005.

```
security = 'IBM US Equity';
field = 'DVD_HIST';                          % Dividend history field
override = {'DVD_START_DT','DVD_END_DT'};  % Dividend start and
                                             % End dates
overridevalues = {'20040101','20050101'};

[d,sec] = getbulkdata(c,security,field,override,overridevalues)
```

```
d =

    DVD_HIST: {{5x7 cell}}

sec =

    'IBM US Equity'
```

`d` is a structure with one field that contains a cell array with the returned bulk data. `sec` contains the IBM security name.

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}
```

```
ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       732246]    [ 732259]    [     732261]    [      732291]    [         0.18]      'Quarter'
    [       732155]    [ 732165]    [     732169]    [      732200]    [         0.18]      'Quarter'
    [       732064]    [ 732073]    [     732077]    [      732108]    [         0.18]      'Quarter'
    [       731973]    [ 731983]    [     731987]    [      732016]    [         0.16]      'Quarter'

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

### Return Specific Field Using Name-Value Pair Arguments

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the closing price and dividend history for IBM with dividend dates from January 1, 2004 through January 1, 2005. Specify the data return format as a character vector by setting the name-value pair argument `'returnFormattedValue'` to `'true'`.

```
security = 'IBM US Equity';
fields = {'LAST_PRICE','DVD_HIST'};        % Closing price and
                                           % Dividend history fields
override = {'DVD_START_DT','DVD_END_DT'};  % Dividend start and
                                           % End dates
overridevalues = {'20040101','20050101'};

[d,sec] = getbulkdata(c,security,fields,override,overridevalues,...
                      'returnFormattedValue',true)
```

```
d =

      DVD_HIST: {{5x7 cell}}
    LAST_PRICE: {'188.74'}

sec =
```

```
    'IBM US Equity'
```

`d` is a structure with two fields. The first field `DVD_HIST` contains a cell array with the dividend historical data as a cell array. The second field `LAST_PRICE` contains a cell array with the closing price as a character vector. `sec` contains the IBM security name.

Display the closing price.

```
d.LAST_PRICE
```

```
ans =

    '188.74'
```

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}
```

```
ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       732246]    [ 732259]    [     732261]    [      732291]    [         0.18]      'Quarter'
    [       732155]    [ 732165]    [     732169]    [      732200]    [         0.18]      'Quarter'
    [       732064]    [ 732073]    [     732077]    [      732108]    [         0.18]      'Quarter'
    [       731973]    [ 731983]    [     731987]    [      732016]    [         0.16]      'Quarter'

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

### Return Bulk Data as Table with Datetime

Create a Bloomberg® connection, and then request dividend history data. The `getbulkdata` function returns data for dates as a `datetime` array.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `getbulkdata` function returns data as a structure.

Return dates as a `datetime` array by setting the `DatetimeType` property of the connection object. In this case, the table contains dates in variables that are `datetime` arrays.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Return the dividend history for IBM®.

```
s = 'IBM US Equity';
f = 'DVD_HIST'; % Dividend history field

d = getbulkdata(c,s,f);
```

Display the first three rows of the table.

```
d.DVD_HIST{1}(1:3,:)
```

```
ans =

  3×7 table

       DeclaredDate            ExmDate             RecordDate            PayableDate
    ____________________    ____________________    ____________________    ____________

    31-Oct-2017 00:00:00    09-Nov-2017 00:00:00    10-Nov-2017 00:00:00    09-Dec-2017 00:00:00
    25-Jul-2017 00:00:00    08-Aug-2017 00:00:00    10-Aug-2017 00:00:00    09-Sep-2017 00:00:00
    25-Apr-2017 00:00:00    08-May-2017 00:00:00    10-May-2017 00:00:00    10-Jun-2017 00:00:00
```

Display three declared dates. The `DeclaredDate` variable is a `datetime` array.

```
d.DVD_HIST{1}.DeclaredDate(1:3)
```

```
ans =

  3×1 datetime array

   31-Oct-2017 00:00:00
   25-Jul-2017 00:00:00
   25-Apr-2017 00:00:00
```

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### s — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector or string scalar for one security or a cell array of character vectors or string array for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell` | `string`

**`f` — Bloomberg data fields**
character vector | string scalar | cell array of character vectors | string array

Bloomberg data fields, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg data field name. A cell array of character vectors or string array denotes multiple Bloomberg data field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `{'LAST_PRICE';'OPEN'}`

Data Types: `char` | `cell` | `string`

**`o` — Bloomberg override field**
`[]` (default) | character vector | string scalar | cell array of character vectors | string array

Bloomberg override field, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg override field name. A cell array of character vectors or string array denotes multiple Bloomberg override field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `'END_DT'`

Data Types: `char` | `cell` | `string`

**`ov` — Bloomberg override field value**
`[]` (default) | character vector | string scalar | cell array of character vectors | string array

Bloomberg override field value, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg override field value. A cell array of character vectors or string array denotes multiple Bloomberg override field values. Use this field value to filter the Bloomberg data result set.

Example: `'20100101'`

Data Types: `char` | `cell` | `string`

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'returnFormattedValue',true`

**`returnEids` — Entitlement identifiers**
true | false

Entitlement identifiers, specified as the comma-separated pair consisting of `'returnEids'` and a Boolean. `true` adds a name and value for the entitlement identifier (EID) date to the return data.

Data Types: `logical`

**`returnFormattedValue` — Return format**
true | false

Return format, specified as the comma-separated pair consisting of `'returnFormattedValue'` and a Boolean. `true` forces all data to be returned as the data type character vector.

Data Types: `logical`

**`useUTCTime` — Date time format**
true | false

Date time format, specified as the comma-separated pair consisting of `'useUTCTime'` and a Boolean. `true` returns date and time values as Coordinated Universal Time (UTC) and `false` defaults to the Bloomberg **TZDF <GO>** settings of the requestor.

Data Types: `logical`

**`forcedDelay` — Latest reference data**
true | false

Latest reference data, specified as the comma-separated pair consisting of `'forcedDelay'` and a Boolean. `true` returns the latest data up to the delay period specified by the exchange for the security.

Data Types: `logical`

## Output Arguments

**`d` — Bloomberg data**
structure (default) | table | timetable

Bloomberg data, returned as a structure, table, or timetable. The data type of the Bloomberg data depends on the DataReturnFormat and DatetimeType properties of the connection object. For details about the data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**`sec` — Security list**
cell array of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`

- `svm`
- `ticker` (default)
- `wpk`

## See Also

`blp` | `close` | `getdata` | `history` | `realtime` | `timeseries`

**Topics**

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2014b**

# getdata

Current data for Bloomberg connection V3

## Syntax

```
d = getdata(c,s,f)
d = getdata(c,s,f,o,ov)
d = getdata(c,s,f,o,ov,Name,Value)
[d,sec] = getdata( ___ )
```

## Description

`d = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`. `getdata` accesses the Bloomberg reference data service.

`d = getdata(c,s,f,o,ov)` returns the data using the override fields `o` with corresponding override values `ov`.

`d = getdata(c,s,f,o,ov,Name,Value)` returns the data using name-value pair arguments for additional Bloomberg request settings.

`[d,sec] = getdata( ___ )` additionally returns the security list `sec` using any of the input argument combinations in the previous syntaxes.

## Examples

### Last and Open Price for Security

First, create a Bloomberg Desktop connection. Then, request last and open pries for a security. The current data you see when running this code can differ from the output data here.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Request last and open prices for Microsoft.

```
[d,sec] = getdata(c,'MSFT US Equity',{'LAST_PRICE';'OPEN'})
```

```
d =
    LAST_PRICE: 33.3401
          OPEN: 33.6000

sec =
    'MSFT US Equity'
```

`getdata` returns a structure `d` with the last and open prices. Also, `getdata` returns the security in `sec`.

Close the connection.

```
close(c)
```

### Specified Fields Given Override Fields and Values

First, create a Bloomberg Desktop connection. Then, request data for specific fields for a security using an override field and value. The current data you see when running this code can differ from the output data here.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Request data for Bloomberg fields `'YLD_YTM_ASK'`, `'ASK'`, and `'OAS_SPREAD_ASK'` when the Bloomberg field `'OAS_VOL_ASK'` is `'14.000000'`.

```
[d,sec] = getdata(c,'030096AF8 Corp',...
   {'YLD_YTM_ASK','ASK','OAS_SPREAD_ASK','OAS_VOL_ASK'},...
   {'OAS_VOL_ASK'},{'14.000000'})
```

```
d =
       YLD_YTM_ASK: 5.6763
               ASK: 120.7500
    OAS_SPREAD_ASK: 307.9824
       OAS_VOL_ASK: 14

sec =
    '030096AF8 Corp'
```

`getdata` returns a structure `d` with the resulting values for the requested fields.

Close the connection.

```
close(c)
```

### Request for Security Using CUSIP Number

First, create a Bloomberg Desktop connection. Then, use the CUSIP number for a security to request last price. The current data you see when running this code can differ from the output data here.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Request the last price for IBM with the CUSIP number.

```
d = getdata(c,'/cusip/459200101','LAST_PRICE')
```

```
d =
    LAST_PRICE: 182.5100
```

`getdata` returns a structure `d` with the last price.

Close the connection.

```
close(c)
```

### Last Price for Security with Pricing Source

First, create a Bloomberg Desktop connection. Then, request the last price for a security. Specify the security using the CUSIP number with a pricing source. The current data you see when running this code can differ from the output data here.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Specify IBM with the CUSIP number and the pricing source `BGN` after the `@` symbol.

```
d = getdata(c,'/cusip/459200101@BGN','LAST_PRICE')
```

```
d =
    LAST_PRICE: 186.81
```

`getdata` returns a structure `d` with the last price.

Close the connection.

```
close(c)
```

### Constituent Weights Using Date Override

First, create a Bloomberg Desktop connection. Then, request the constituent weights of an index using a date override. The current data you see when running this code can differ from the output data here.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the constituent weights for the Dow Jones Index as of January 1, 2010 using a date override with the required date format `YYYYMMDD`.

```
d = getdata(c,'DJX Index','INDX_MWEIGHT','END_DT','20100101')
```

```
d =
    INDX_MWEIGHT: {{30x2 cell}}
```

`getdata` returns a structure `d` with a cell array where the first column is the index and the second column is the constituent weight.

Display the constituent weights for each index.

```
d.INDX_MWEIGHT{1,1}
```

```
ans =
    'AA UN'      [1.1683]
    'AXP UN'     [2.9366]
    'BA UN'      [3.9229]
    'BAC UN'     [1.0914]
    ...
```

Close the connection.

```
close(c)
```

### Current Data and Dates as Table with Datetime

Create a Bloomberg® connection, and then request current data for specific fields. The `getdata` function returns data for dates as a `datetime` array.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `getdata` function returns data as a structure.

Return dates as a `datetime` array by setting the `DatetimeType` property of the connection object. In this case, the table contains dates in variables that are `datetime` arrays.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Request current data for these fields:

- Last update date
- Last price
- Number of trades
- Previous real-time trading date

```
s = 'IBM US Equity';
f = {'LAST_UPDATE_DT','LAST_PRICE', ...
    'NUM_TRADES_RT','PREV_TRADING_DT_REALTIME'};
d = getdata(c,s,f)
```

```
d =

  1×4 table
```

```
      LAST_UPDATE_DT       LAST_PRICE    NUM_TRADES_RT    PREV_TRADING_DT_REALTIME
    ___________________    __________    _____________    ________________________

    21-Dec-2017 00:00:00      152.2          24846          20-Dec-2017 00:00:00
```

Display the last update date. This date is a `datetime` array.

```
d.LAST_UPDATE_DT
```

```
ans =

  datetime

   21-Dec-2017 00:00:00
```

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### s — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector or string scalar for one security or a cell array of character vectors or string array for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell` | `string`

### f — Bloomberg data fields

character vector | string scalar | cell array of character vectors | string array

Bloomberg data fields, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg data field name. A cell array of character vectors or string array denotes multiple Bloomberg data field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `{'LAST_PRICE';'OPEN'}`

Data Types: `char` | `cell` | `string`

### o — Bloomberg override field

`[]` (default) | character vector | string scalar | cell array of character vectors | string array

Bloomberg override field, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg override field name. A cell array

of character vectors or string array denotes multiple Bloomberg override field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `'END_DT'`

Data Types: `char` | `cell` | `string`

**`ov` — Bloomberg override field value**
`[]` (default) | character vector | string scalar | cell array of character vectors | string array

Bloomberg override field value, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg override field value. A cell array of character vectors or string array denotes multiple Bloomberg override field values. Use this field value to filter the Bloomberg data result set.

Example: `'20100101'`

Data Types: `char` | `cell` | `string`

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'returnEids',true`

**`returnEids` — Entitlement identifiers**
true | false

Entitlement identifiers, specified as the comma-separated pair consisting of `'returnEids'` and a Boolean. `true` adds a name and value for the entitlement identifier (EID) date to the return data.

Data Types: `logical`

**`returnFormattedValue` — Return format**
true | false

Return format, specified as the comma-separated pair consisting of `'returnFormattedValue'` and a Boolean. `true` forces all data to be returned as the data type character vector.

Data Types: `logical`

**`useUTCTime` — Date time format**
true | false

Date time format, specified as the comma-separated pair consisting of `'useUTCTime'` and a Boolean. `true` returns date and time values as Coordinated Universal Time (UTC) and `false` defaults to the Bloomberg **TZDF <GO>** settings of the requestor.

Data Types: `logical`

**`forcedDelay` — Latest reference data**
true | false

Latest reference data, specified as the comma-separated pair consisting of `'forcedDelay'` and a Boolean. `true` returns the latest data up to the delay period specified by the exchange for the security.

Data Types: `logical`

## Output Arguments

### `d` — Bloomberg data

structure (default) | table | timetable

Bloomberg data, returned as a structure, table, or timetable. The data type of the Bloomberg data depends on the DataReturnFormat and DatetimeType properties of the connection object. For details about the data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

### `sec` — Security list

cell array of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`
- `svm`
- `ticker` (default)
- `wpk`

## Tips

- Bloomberg V3 data supports additional name-value pair arguments. To access further information on these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.
- You can check data and field availability by using the Bloomberg Excel® Add-In.
- For a Bloomberg B-PIPE connection, `d` returns an additional field named `EID`. `EID` means entitlement identifier. For details, see the *Bloomberg API Developer's Guide*.

## See Also

`blp` | `close` | `history` | `realtime` | `timeseries`

### Topics

"Retrieve Bloomberg Current Data" on page 3-6
"Retrieve Current and Historical Data Using Bloomberg" on page 1-14

“Workflow for Bloomberg” on page 3-19

**Introduced in R2010a**

# history

Historical data for Bloomberg connection V3

## Syntax

```
d = history(c,s,f,fromdate,todate)
d = history(c,s,f,fromdate,todate,period)
d = history(c,s,f,fromdate,todate,period,currency)
d = history(c,s,f,fromdate,todate,period,currency,Name,Value)
[d,sec] = history( ___ )
```

## Description

`d = history(c,s,f,fromdate,todate)` returns the historical data for the security list `s` and the connection object `c` for the fields `f` for the dates `fromdate` through `todate`. Date strings can be input in any format recognized by MATLAB. `sec` is the security list that maps the order of the return data. The return data `d` is sorted to match the input order of `s`.

`d = history(c,s,f,fromdate,todate,period)` returns the historical data for the fields `f` and the dates `fromdate` through `todate` with a specific periodicity `period`.

`d = history(c,s,f,fromdate,todate,period,currency)` returns the historical data for the security list `s` for the fields `f` and the dates `fromdate` through `todate` based on the given currency `currency`.

`d = history(c,s,f,fromdate,todate,period,currency,Name,Value)` returns the historical data for the security list `s` using additional options specified by one or more name-value pair arguments.

`[d,sec] = history( ___ )` additionally returns the security list `sec` using any of the input argument combinations in the previous syntaxes. The return data, `d` and `sec`, are sorted to match the input order of `s`.

## Examples

### Daily Closing Prices Within Date Range

First, create a Bloomberg Desktop connection. Then, retrieve the daily closing price for a security within a date range.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the daily closing price from August 1, 2010 through August 10, 2010 for the IBM security.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','8/10/2010')

d =

     734352.00        123.55
     734353.00        123.18
     734354.00        124.03
     734355.00        124.56
     734356.00        123.58
     734359.00        125.34
     734360.00        125.19


sec =

    'IBM US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Monthly Closing Prices Within Date Range

First, create a Bloomberg Desktop connection. Then, retrieve the monthly closing prices for a security within a date range.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the monthly closing price from August 1, 2010 through December 10, 2010 for the IBM security.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','12/10/2010','monthly')

d =

     734360.00        125.19
     734391.00        121.53
     734421.00        131.85
     734452.00        139.78
     734482.00        138.13


sec =

    'IBM US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Monthly Closing Prices Within Date Range Using US Currency

First, create a Bloomberg Desktop connection. Then, retrieve the monthly closing prices for a security within a date range. Specify prices using the US currency.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the monthly closing price from August 1, 2010 through December 10, 2010 for the IBM security in US currency `'USD'`.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','12/10/2010','monthly','USD')
```

```
d =

     734360.00        125.19
     734391.00        121.53
     734421.00        131.85
     734452.00        139.78
     734482.00        138.13


sec =

    'IBM US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Monthly Closing Prices Within Date Range Using Currency with Specified Period

First, create a Bloomberg Desktop connection. Then, retrieve the monthly closing prices for a security within a date range. Specify prices using the US currency. Specify period values to customize the returned data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the monthly closing price from August 1, 2010 through August 1, 2011 for the IBM security in US currency. The period values `'monthly'`, `'actual'`, and `'all_calendar_days'` specify returning actual monthly data for all calendar days. The period value `'nil_value'` specifies filling missing data values with a `NaN`.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','8/01/2011',{'monthly','actual',...
                  'all_calendar_days','nil_value'},'USD')
```

```
d =

     734351.00        128.40
     734382.00        125.77
     734412.00        135.64
     734443.00        143.32
     734473.00        144.41
     734504.00        146.76
     734535.00        163.56
     734563.00        159.97
     734594.00        164.27
     734624.00        170.58
     734655.00        166.56
     734685.00        174.54
     734716.00        180.75


sec =

    'IBM US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Daily Closing Prices Within Date Range Using Currency with Name-Value Pairs

First, create a Bloomberg Desktop connection. Then, retrieve the daily closing prices for a security within a date range. Specify prices using the US currency. Use name-value pair arguments to adjust the prices.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the daily closing price from August 1, 2010 through August 10, 2010 for the IBM security in U.S. currency `'USD'`. The prices are adjusted for normal cash and splits.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','8/10/2010','daily','USD',...
```

```
                'adjustmentNormal',true,...
                'adjustmentSplit',true)

d =

     734352.00        123.55
     734353.00        123.18
     734354.00        124.03
     734355.00        124.56
     734356.00        123.58
     734359.00        125.34
     734360.00        125.19


sec =

    'IBM US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Daily Closing Prices Using CUSIP Number and Pricing Source

First, create a Bloomberg Desktop connection. Then, retrieve the daily closing prices for a security within a date range. Specify the security using the CUSIP number and a pricing source.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the daily closing price from January 1, 2012 through January 1, 2013 for the security specified with a CUSIP number `/cusip/459200101` and with pricing source `BGN`.

`d` contains the numeric representation for the date in the first column and the closing price in the second column.

```
d = history(c,'/cusip/459200101@BGN','LAST_PRICE',...
            '01/01/2012','01/01/2013')

d =

     734871.00        180.69
     734872.00        179.96
     734873.00        179.10
     ...
```

Close the Bloomberg connection.

```
close(c)
```

### Closing Prices Within Date Range Using International Date Format

First, create a Bloomberg Desktop connection. Then, retrieve the closing prices for a security within a date range. Specify the dates for the range using an international date format.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the closing price for the given dates in international format for the security `'MSFT@BGN US Equity'`.

```
stDt = datenum('01/06/11','dd/mm/yyyy');
endDt = datenum('01/06/12','dd/mm/yyyy');
[d,sec] = history(c,'MSFT@BGN US Equity','LAST_PRICE',...
                  stDt,endDt,{'previous_value','all_calendar_days'})
```

```
d =

     734655.00         22.92
     734656.00         22.72
     734657.00         22.42
     ...

sec =

    'MSFT@BGN US Equity'
```

`d` contains the numeric representation for the date in the first column and the closing price in the second column. `sec` contains the name of the IBM security.

Close the Bloomberg connection.

```
close(c)
```

### Median Estimated Earnings Per Share Using Override Fields

First, create a Bloomberg Desktop connection. Then, retrieve the median earnings per share for a security within a date range. Specify an override field and value.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the median estimated earnings per share for AkzoNobel® from October 1, 2010 through October 30, 2010. When specifying Bloomberg override fields, use the character vector

'overrideFields'. The overrideFields argument must be an n-by-2 cell array, where the first column is the override field and the second column is the override value.

```
d = history(c,'AKZA NA Equity', ...
            'BEST_EPS_MEDIAN',datenum('01.10.2010', ...
            'dd.mm.yyyy'),datenum('30.10.2010','dd.mm.yyyy'), ...
            {'daily','calendar'},[],'overrideFields', ...
            {'BEST_FPERIOD_OVERRIDE','BF'})
```

```
d =

     734412.00          3.75
     734415.00          3.75
     734416.00          3.75
     ...
```

d returns the numeric representation for the date in the first column and the median estimated earnings per share in the second column.

Close the Bloomberg connection.

```
close(c)
```

### Historical Data as Table with Dates

Create a Bloomberg® connection, and then retrieve closing prices for a historical date range. The history function returns data for dates as a datetime array.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using blpsrv or Bloomberg B-PIPE® using bpipe.

Return data as a table by setting the DataReturnFormat property of the connection object. If you do not set this property, the history function returns data as a numeric array.

Return dates as a datetime array by setting the DatetimeType property of the connection object. In this case, the table contains dates in variables that are datetime arrays.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Retrieve historical closing prices for IBM® from August 1, 2010 through August 10, 2010. d is a table that contains dates as a datetime array.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE', ...
    '8/01/2010','8/10/2010')
```

```
d =
```

```
  7×2 table

      DATE        LAST_PRICE
    ___________    __________

    02-Aug-2010      130.76
    03-Aug-2010      130.37
    04-Aug-2010      131.27
    05-Aug-2010      131.83
    06-Aug-2010      130.14
    09-Aug-2010      132.00
    10-Aug-2010      131.84


sec =

  1×1 cell array

    {'IBM US Equity'}
```

Access dates in the returned data.

```
d.DATE
```

```
ans =

  7×1 datetime array

   02-Aug-2010
   03-Aug-2010
   04-Aug-2010
   05-Aug-2010
   06-Aug-2010
   09-Aug-2010
   10-Aug-2010
```

Close the Bloomberg connection.

```
close(c)
```

### Historical Data as Timetable

Create a Bloomberg® connection, and then retrieve closing prices for a historical date range. The `history` function returns data as a `timetable`.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a `timetable` by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `history` function returns data as a numeric array.

```
c.DataReturnFormat = 'timetable';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Retrieve historical closing prices for IBM® from August 1, 2010 through August 10, 2010. `d` is a `timetable` that contains dates in the first column.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE', ...
    '8/01/2010','8/10/2010')
```

```
d =

  7×1 timetable

       DATE        LAST_PRICE
    ____________    __________

    02-Aug-2010       130.76
    03-Aug-2010       130.37
    04-Aug-2010       131.27
    05-Aug-2010       131.83
    06-Aug-2010       130.14
    09-Aug-2010       132.00
    10-Aug-2010       131.84


sec =

  1×1 cell array

    {'IBM US Equity'}
```

Access dates in the returned data.

```
d.DATE
```

```
ans =

  7×1 datetime array

   02-Aug-2010
   03-Aug-2010
   04-Aug-2010
   05-Aug-2010
   06-Aug-2010
   09-Aug-2010
   10-Aug-2010
```

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### s — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector or string scalar for one security or a cell array of character vectors or string array for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell` | `string`

### f — Bloomberg data fields

character vector | string scalar | cell array of character vectors | string array

Bloomberg data fields, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg data field name. A cell array of character vectors or string array denotes multiple Bloomberg data field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `{'LAST_PRICE';'OPEN'}`

Data Types: `char` | `cell` | `string`

### period — Periodicity

`'daily'` | `'weekly'` | `'monthly'` | `'quarterly'` | ...

Periodicity, specified as one of these values to denote the data to return. For specifying multiple values, use a cell array. For example, when `period` is set to `{'daily','all_calendar_days'}`, `history` returns daily data for all calendar days, and reports missing data as `NaN`s. When `period` is set to `'active_days_only'`, `history` returns data using the default periodicity for active trading days only. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. These tables show the values for `period`.

To specify the periodicity of the return data, see this table.

| Value | Description |
|---|---|
| `'daily'` | Return data for each day. |
| `'weekly'` | Return data for each week. |
| `'monthly'` | Return data for each month. |
| `'quarterly'` | Return data for each quarter. |
| `'semi_annually'` | Return data semiannually. |
| `'yearly'` | Return data for each year. |

The anchor date is the date to which all other reported dates are related. To specify the anchor date, see this table.

| Value | Description |
|---|---|
| `'actual'` | Anchor date specification for an actual date. For this function, for periodicities other than daily, `todate` is the anchor date.<br><br>If the period is weekly and the `todate` is a Thursday, every data point is a Thursday, or the nearest prior business day to Thursday. If the period is monthly and the `todate` is the 20th of a month, every data point is the 20th of each month in the date range. |
| `'calendar'` | Anchor date specification for a calendar year. |
| `'fiscal'` | Anchor date specification for a fiscal year. |
| `'none'` | Do not specify the anchor date. |

To specify returning data for particular days, see this table.

| Value | Description |
|---|---|
| `'non_trading_weekdays'` | Return data for all weekdays. |
| `'all_calendar_days'` | Return data for all calendar days. |
| `'active_days_only'` | Return data for only active trading days. |

To specify how to fill missing values, see this table.

| Value | Description |
|---|---|
| `'previous_value'` | Fill missing values with previous values for dates without trading activity for the security. If no previous value exists in the month before the `fromdate`, this function retains the missing values. |
| `'nil_value'` | Fill missing values with a `NaN` for dates without trading activity for the security. |

Data Types: `char` | `cell`

### `currency` — Currency

character vector | string scalar

Currency, specified as a character vector or string scalar to denote the ISO® code for the currency of the returned data. For example, to specify output money values in U.S. currency, use `USD` for this argument.

Data Types: `char` | `string`

### `fromdate` — Beginning date

double scalar | character vector | string scalar | datetime

Beginning date for the historical data, specified as a double scalar, character vector, string scalar, or `datetime` array. You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

Data Types: `datetime` | `double` | `char` | `string`

**`todate` — End date**
double scalar | character vector | string scalar | datetime

End date for the historical data, specified as a double scalar, character vector, string scalar, or `datetime` array. You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

Data Types: `datetime` | `double` | `char` | `string`

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'adjustmentNormal',true`

**`overrideFields` — Override fields**
cell array

Override fields, specified as the comma-separated pair consisting of `'overrideFields'` and an `n`-by-2 cell array. The first column of the cell array is the override field and the second column is the override value.

Example: `'overrideFields',
{'IVOL_DELTA_LEVEL','DELTA_LVL_10';'IVOL_DELTA_PUT_OR_CALL','IVOL_PUT';'IVOL_MATURITY','MATURITY_1STM'}`

Data Types: `cell`

**`adjustmentNormal` — Historical normal pricing adjustment**
true | false

Historical normal pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentNormal'` and a Boolean to reflect:

- Regular Cash
- Interim
- 1st Interim
- 2nd Interim
- 3rd Interim
- 4th Interim
- 5th Interim
- Income
- Estimated
- Partnership Distribution
- Final
- Interest on Capital
- Distribution

- Prorated

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

**`adjustmentAbnormal` — Historical abnormal pricing adjustment**
true | false

Historical abnormal pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentAbnormal'` and a Boolean to reflect:

- Special Cash
- Liquidation
- Capital Gains
- Long-Term Capital Gains
- Short-Term Capital Gains
- Memorial
- Return of Capital
- Rights Redemption
- Miscellaneous
- Return Premium
- Preferred Rights Redemption
- Proceeds/Rights
- Proceeds/Shares
- Proceeds/Warrants

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

**`adjustmentSplit` — Historical split pricing or volume adjustment**
true | false

Historical split pricing or volume adjustment, specified as the comma-separated pair consisting of `'adjustmentSplit'` and a Boolean to reflect:

- Spin-Offs
- Stock Splits/Consolidations
- Stock Dividend/Bonus
- Rights Offerings/Entitlement

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

**`adjustmentFollowDPDF` — Historical pricing adjustment**
true (default) | false

Historical pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentFollowDPDF'` and a Boolean. Setting this name-value pair follows the **DPDF <GO>** option from the Bloomberg terminal. For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

## Output Arguments

### d — Bloomberg historical data

numeric array (default) | table | timetable

Bloomberg historical data, returned as a numeric array, table, or timetable. The data type of the historical data depends on the DataReturnFormat and DatetimeType properties of the connection object. The first column (or field) in the historical data contains the date. The remaining columns contain the requested data fields.

For details about the data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

### sec — Security list

cell array of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`
- `svm`
- `ticker` (default)
- `wpk`

## Tips

- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/javaclasspath.txt`. For details about the static Java class path, see "Static Path".
- You can check data and field availability by using the Bloomberg Excel Add-In.

## See Also

`blp` | `close` | `getdata` | `realtime` | `timeseries`

**Topics**
“Retrieve Bloomberg Historical Data” on page 3-8
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010a**

# isconnection

Determine Bloomberg connection V3

## Syntax

```
v = isconnection(c)
```

## Description

`v = isconnection(c)` returns `true` if `c` is a valid Bloomberg V3 connection and `false` otherwise.

## Examples

### Validate the Bloomberg Connection

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Validate the Bloomberg connection.

```
v = isconnection(c)
```

```
v =

     1
```

`v` returns `true` showing that the Bloomberg connection is valid.

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

## Output Arguments

### v — Valid Bloomberg connection

true | false

Valid Bloomberg connection, returned as a logical true, 1, or a logical false, 0.

## See Also

`blp` | `blpsrv` | `bpipe` | `close` | `getdata`

**Topics**

“Connect to Bloomberg” on page 3-2
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010b**

# lookup

Find information about securities for Bloomberg connection V3

## Syntax

```
l = lookup(c,q,reqtype,Name,Value)
```

## Description

`l = lookup(c,q,reqtype,Name,Value)` retrieves data based on criteria in the query `q` for a specific request type `reqtype` using the Bloomberg connection `c`. For additional information about the query criteria and the possible name-value pair combinations, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

## Examples

### Look Up Security

Create a Bloomberg® connection, and then use the Security Lookup to retrieve information about the IBM® corporate bond. For details about Bloomberg and the parameter values you can set, see the *Bloomberg API Developer's Guide* using the **WAPI<GO>** option from the Bloomberg terminal.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `lookup` function returns data as a structure.

```
c.DataReturnFormat = 'table';
```

Retrieve the instrument data for an IBM corporate bond with a maximum of 20 rows of data. The Security Lookup returns the security names and descriptions.

```
insts = lookup(c,'IBM','instrumentListRequest','maxResults',20, ...
    'yellowKeyFilter','YK_FILTER_CORP', ...
    'languageOverride','LANG_OVERRIDE_NONE');
```

Display the first three rows in the table. The first column contains the IBM corporate bond names, and the second column contains the bond descriptions.

```
insts(1:3,:)
```

```
ans =

  3×2 table
```

```
        security                    description
    ____________________    ____________________________________

    'DD103619 <corp>'    'International Business Machines Corp'
    '459200AG <corp>'    'International Business Machines Corp'
    'EC767659 <corp>'    'International Business Machines Corp'
```

Close the Bloomberg connection.

```
close(c)
```

### Look Up Curve

Use the Curve Lookup to retrieve information about the `'GOLD'` related curve `'CD1016'`. For details about Bloomberg and the parameter values you can set, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Connect to Bloomberg.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the curve data for the credit default swap subtype of corporate bonds for a `'GOLD'` related curve `'CD1016'`. Return a maximum of `10` rows of data for the U.S. with `'USD'` currency.

```
curves = lookup(c,'GOLD','curveListRequest','maxResults',10,...
                'countryCode','US','currencyCode','USD',...
                'curveid','CD1016','type','CORP','subtype','CDS')
```

```
curves =

            curve: {'YCCD1016 Index'}
      description: {'Goldman Sachs Group Inc/The'}
          country: {'US'}
         currency: {'USD'}
          curveid: {'CD1016'}
             type: {'CORP'}
          subtype: {'CDS'}
        publisher: {'Bloomberg'}
            bbgid: {''}
```

One row of data displays as Bloomberg curve name `'YCCD1016 Index'` with Bloomberg description `'Goldman Sachs Group Inc/The'` in the U.S. with `'USD'` currency. The Bloomberg short-form identifier for the curve is `'CD1016'`. Bloomberg is the publisher and the `bbgid` is blank.

Close the Bloomberg connection.

```
close(c)
```

### Look Up Government Security

Use the Government Security Lookup to retrieve information for United States Treasury bonds. For details about Bloomberg and the parameter values you can set, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Connect to Bloomberg.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Filter government security data with ticker filter of `'T'` for a maximum of `10` rows of data.

```
govts = lookup(c,'T','govtListRequest','maxResults',10,...
               'partialMatch',false)
```

```
govts =

      parseky: {10x1 cell}
         name: {10x1 cell}
       ticker: {10x1 cell}
```

The Government Security Lookup returns `parseky` data, the name, and ticker of the United States Treasury bonds.

Display the `parseky` data.

```
govts.parseky
```

```
ans =
    '912828VS Govt'
    '912828RE Govt'
    '912810RC Govt'
    '912810RB Govt'
    '912828VU Govt'
    '912828VV Govt'
    '912828VB Govt'
    '912828VR Govt'
    '912828VW Govt'
    '912828VQ Govt'
```

Display the names of the United States Treasury bonds.

```
govts.name
```

```
ans =
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
```

Display the tickers of the United States Treasury bonds.

```
govts.ticker
```

```
ans =
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
```

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `q` — Keyword query

character vector | string scalar | cell array of character vectors | string array

Keyword query, specified as a character vector, string scalar, cell array of character vectors, or string array. Each character vector or string denotes an item for which information is requested. For example, the keyword query can be a security, a curve type, or a filter ticker.

Data Types: `char` | `cell` | `string`

### `reqtype` — Request type

`'instrumentListRequest'` | `'curveListRequest'` | `'govtListRequest'`

Request type, specified as the preceding values to denote the type of information request. `'instrumentListRequest'` denotes a security or instrument lookup request. `'curveListRequest'` denotes a curve lookup request. `'govtListRequest'` denotes a government lookup request for government securities.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'maxResults'`, `20`, `'yellowKeyFilter'`, `'YK_FILTER_CORP'`, `'languageOverride'`, `'LANG_OVERRIDE_NONE'`, `'countryCode'`, `'US'`, `'currencyCode'`, `'USD'`, `'curveid'`, `'CD1016'`, `'type'`, `'CORP'`, `'subtype'`, `'CDS'`, `'partialMatch'`, `false`

### `maxResults` — Number of rows in result data

numeric scalar

Number of rows in the result data, specified as the comma-separated pair consisting of `'maxResults'` and a numeric scalar to denote the total maximum number of rows of information to return. Result data can be one or more rows of data no greater than the number specified.

Data Types: `double`

**`yellowKeyFilter` — Bloomberg yellow key filter**
character vector | string scalar

Bloomberg yellow key filter, specified as the comma-separated pair consisting of `'yellowKeyFilter'` and a unique character vector or string scalar to denote the particular yellow key for government securities, corporate bonds, equities, and commodities, for example.

Data Types: `char` | `string`

**`languageOverride` — Language override**
character vector | string scalar

Language override, specified as the comma-separated pair consisting of `'languageOverride'` and a unique character vector or string scalar to denote a translation language for the result data.

Data Types: `char` | `string`

**`countryCode` — Country code**
character vector | string scalar

Country code, specified as the comma-separated pair consisting of `'countryCode'` and a character vector or string scalar to denote the country for the result data.

Data Types: `char` | `string`

**`currencyCode` — Currency code**
character vector | string scalar

Currency code, specified as the comma-separated pair consisting of `'currencyCode'` and a character vector or string scalar to denote the currency for the result data.

Data Types: `char` | `string`

**`curveID` — Bloomberg short-form identifier for curve**
character vector | string scalar

Bloomberg short-form identifier for a curve, specified as the comma-separated pair consisting of `'curveID'` and a character vector or string scalar.

Data Types: `char` | `string`

**`type` — Bloomberg market sector type**
character vector | string scalar

Bloomberg market sector type corresponding to the Bloomberg yellow keys, specified as the comma-separated pair consisting of `'type'` and a character vector or string scalar.

Data Types: `char` | `string`

**`subtype` — Bloomberg market sector subtype**
character vector | string scalar

Bloomberg market sector subtype, specified as the comma-separated pair consisting of `'subtype'` and a character vector or string scalar to further delineate the market sector type.

Data Types: `char` | `string`

**`partialMatch` — Partial match on ticker**
`true` | `false`

Partial match on ticker, specified as the comma-separated pair consisting of `'partialMatch'` and `true` or `false`. When set to `true`, you can filter securities by setting `q` to a query such as `'T*'`. When set to `false`, the securities are unfiltered.

Data Types: `logical`

## Output Arguments

**`l` — Lookup information**
structure (default) | table

Lookup information, returned as a structure or table containing set properties depending on the request type. The data type of the lookup information depends on the DataReturnFormat property of the connection object.

For a list of the set properties and their descriptions, see the following tables.

**'instrumentListRequest' Properties**

| Property | Description |
|---|---|
| `security` | Security name |
| `description` | Security long name |

**'curveListRequest' Properties**

| Property | Description |
|---|---|
| curve | Bloomberg curve name |
| description | Bloomberg description |
| country | Country code |
| currency | Currency code |
| curveid | Bloomberg short-form identifier for the curve |
| type | Bloomberg market sector type |
| subtype | Bloomberg market sector subtype |
| publisher | Bloomberg specified as publisher |
| bbgid | Bloomberg identifier |

**'govtListRequest' Properties**

| Property | Description |
|---|---|
| parsekey | Bloomberg security identifier (ticker or CUSIP, for example), price source, and source key (Bloomberg yellow key) |
| name | Government security name |
| ticker | Government security ticker |

## See Also

blp | close | getdata | history | realtime | timeseries

**Topics**

“Retrieve Bloomberg Current Data” on page 3-6
“Retrieve Current and Historical Data Using Bloomberg” on page 1-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2014a**

# portfolio

Current portfolio data for Bloomberg connection V3

## Syntax

```
d = portfolio(c,p,f)
d = portfolio(c,p,f,o,ov)
[d,plist] = portfolio( ___ )
```

## Description

`d = portfolio(c,p,f)` returns current portfolio data for the fields `f` in the portfolio `p` using the Bloomberg connection `c`.

`d = portfolio(c,p,f,o,ov)` returns current portfolio data using override field `o` and override value `ov`.

`[d,plist] = portfolio( ___ )` also returns the portfolio list `plist` using any of the input argument combinations in the previous syntaxes.

## Examples

### Request Portfolio Data

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Request portfolio data for a custom portfolio with portfolio identifier `U335877-1 Client`. Request data using all fields `f`.

```
p = 'U335877-1 Client';
f = {'PORTFOLIO_MEMBERS','PORTFOLIO_MPOSITION',...
     'PORTFOLIO_MWEIGHT','PORTFOLIO_DATA'};

d = portfolio(c,p,f)
```

```
d =

    PORTFOLIO_MPOSITION: {{0x1 cell}}
      PORTFOLIO_MWEIGHT: {{0x1 cell}}
         PORTFOLIO_DATA: {{0x1 cell}}
      PORTFOLIO_MEMBERS: {{0x1 cell}}
```

`d` is a structure that contains portfolio data. Each structure field corresponds to data for each portfolio field.

Close the connection.

```
close(c)
```

### Request Portfolio Data Using Specific Date

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Request portfolio data for a custom portfolio with portfolio identifier `U335877-1 Client`. Request data using all fields `f`. Filter the portfolio data by specifying the date of November 3, 2014 using the override value `REFERENCE_DATE` equal to `20141103`.

```
p = 'U335877-1 Client';
f = {'PORTFOLIO_MEMBERS','PORTFOLIO_MPOSITION',...
     'PORTFOLIO_MWEIGHT','PORTFOLIO_DATA'};
o = {'REFERENCE_DATE'};
ov = {'20141103'};

[d,plist] = portfolio(c,p,f,o,ov)
```

```
d =

    PORTFOLIO_MPOSITION: {{0x1 cell}}
      PORTFOLIO_MWEIGHT: {{0x1 cell}}
         PORTFOLIO_DATA: {{0x1 cell}}
      PORTFOLIO_MEMBERS: {{0x1 cell}}

plist =

    'U335877-1 Client'
```

`d` is a structure that contains portfolio data. Each structure field corresponds to data for each portfolio field.

`plist` is a cell array that contains the portfolio identifier.

Close the connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### p — Portfolio

character vector | string scalar

Portfolio, specified as a character vector or string scalar. Specify the portfolio by the ID that you can find in the upper-right corner of the portfolio display page. Append the text `' Client'` (without quotes) to the ID. For example, if the ID is `U335877-1`, then specify `'U335877-1 Client'`.

Access the portfolio display page by using the **PRTU<GO>** option from the Bloomberg terminal. For details, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `'U335877-1 Client'`

Data Types: `char` | `cell` | `string`

### f — Portfolio fields

`'PORTFOLIO_DATA'` | `'PORTFOLIO_MEMBERS'` | `'PORTFOLIO_MPOSITION'` | `'PORTFOLIO_MWEIGHT'`

Portfolio fields, specified as one of the preceding values for one field. To specify multiple fields, use a cell array of these values.

| Bloomberg Field Name | Bloomberg Field Description |
|---|---|
| `'PORTFOLIO_DATA'` | Returns a list of the identifiers, positions, market values, cost, cost date, and cost foreign exchange rate of each security in a custom portfolio. |
| `'PORTFOLIO_MEMBERS'` | Returns a list of identifiers for the members of a custom portfolio. |
| `'PORTFOLIO_MPOSITION'` | Returns a list of identifiers and the position for each security in a custom portfolio. |
| `'PORTFOLIO_MWEIGHT'` | Returns a list of identifiers and the percentage weight for each security in a custom portfolio. |

Data Types: `char` | `cell`

### o — Bloomberg override field

character vector | string scalar | cell array of character vectors | string array

Bloomberg override field, specified as a character vector, string scalar, cell array of character vectors, or string array. The Bloomberg value `'REFERENCE_DATE'` denotes returning Bloomberg data for a specific date.

Data Types: `char` | `cell` | `string`

### ov — Bloomberg override field value

`[]` (default) | character vector | string scalar | cell array of character vectors | string array

Bloomberg override field value, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg override field value. A cell array of character vectors or string array denotes multiple Bloomberg override field values. Use this field value to filter the Bloomberg data result set.

Example: `'20100101'`

Data Types: `char` | `cell` | `string`

## Output Arguments

**d — Portfolio data**
structure (default) | table

Portfolio data, returned as a structure or table. The data type of the portfolio data depends on the DataReturnFormat property of the connection object.

**`plist` — Portfolio list**
cell array of character vectors

Portfolio list, returned as a cell array of character vectors for the corresponding portfolio identifiers in `p`. The contents of `plist` are identical in value and order to `p`.

## See Also

`blp` | `close` | `getdata` | `history` | `realtime` | `timeseries`

**Topics**
“Retrieve Bloomberg Current Data” on page 3-6
“Workflow for Bloomberg” on page 3-19

**Introduced in R2015b**

# realtime

Real-time data for Bloomberg connection V3

## Syntax

```
d = realtime(c,s,f)
[subs,t] = realtime(c,s,f,eventhandler)
```

## Description

`d = realtime(c,s,f)` returns the data for the given connection `c`, security list `s`, and requested fields `f`. `realtime` accesses the Bloomberg Market Data service.

`[subs,t] = realtime(c,s,f,eventhandler)` returns the subscription list `subs` and the timer `t` associated with the real-time event handler for the subscription list. Given connection `c`, the `realtime` function subscribes to a security or securities `s` and requests fields `f`, to update in real time while running an event handler `eventhandler`.

## Examples

### Retrieve Data for One Security

Retrieve a snapshot of data for one security only.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume of the IBM security.

```
d = realtime(c,'IBM US Equity',{'Last_Trade','Volume'})
```

```
d =

    LAST_TRADE: '181.76'
        VOLUME: '7277793'
```

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Data for One Security Using the Event Handler `v3stockticker`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3stockticker` that returns Bloomberg stock tick data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume for the IBM security using the event handler `v3stockticker`.

`v3stockticker` requires the input argument `f` of `realtime` to be `'Last_Trade'` , `'Volume'`, or both.

```
[subs,t] = realtime(c,'IBM US Equity',{'Last_Trade','Volume'},...
                    'v3stockticker')
```

```
subs =

com.bloomberglp.blpapi.SubscriptionList@79f07684

   Timer Object: timer-2

   Timer Settings
      ExecutionMode: fixedRate
             Period: 0.05
           BusyMode: drop
            Running: on

   Callbacks
           TimerFcn: 1x4 cell array
           ErrorFcn: ''
           StartFcn: ''
            StopFcn: ''

** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `realtime` returns the stock tick data for the IBM security with the volume and last trade price.

Real-time data continues to display until you execute the `stop` or `close` function.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Data for Multiple Securities Using the Event Handler `v3stockticker`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3stockticker` that returns Bloomberg stock tick data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume for IBM and Ford Motor Company securities.

`v3stockticker` requires the input argument `f` of the `realtime` function to be `'Last_Trade'`, `'Volume'`, or both.

```
[subs,t] = realtime(c,{'IBM US Equity','F US Equity'},...
                    {'Last_Trade','Volume'},'v3stockticker')
```

```
subs =

com.bloomberglp.blpapi.SubscriptionList@6c1066f6


   Timer Object: timer-3

   Timer Settings
      ExecutionMode: fixedRate
              Period: 0.05
           BusyMode: drop
            Running: on

   Callbacks
           TimerFcn: 1x4 cell array
           ErrorFcn: ''
           StartFcn: ''
            StopFcn: ''

** IBM US Equity ** 32433 @ 181.85 29-Oct-2013 15:50:05
** IBM US Equity ** 200 @ 181.85 29-Oct-2013 15:50:05
** IBM US Equity ** 100 @ 181.86 29-Oct-2013 15:50:05
** F US Equity ** 300 @ 17.575 30-Oct-2013 10:14:06
** F US Equity ** 100 @ 17.57 30-Oct-2013 10:14:06
** F US Equity ** 100 @ 17.5725 30-Oct-2013 10:14:06
...
```

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `realtime` returns the stock tick data for the IBM and Ford Motor Company securities with the last trade price and volume.

Real-time data continues to display until you use the `stop` or `close` function.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Data for One Security Using the Event Handler `v3showtrades`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3showtrades` that creates a figure showing requested data for a security.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

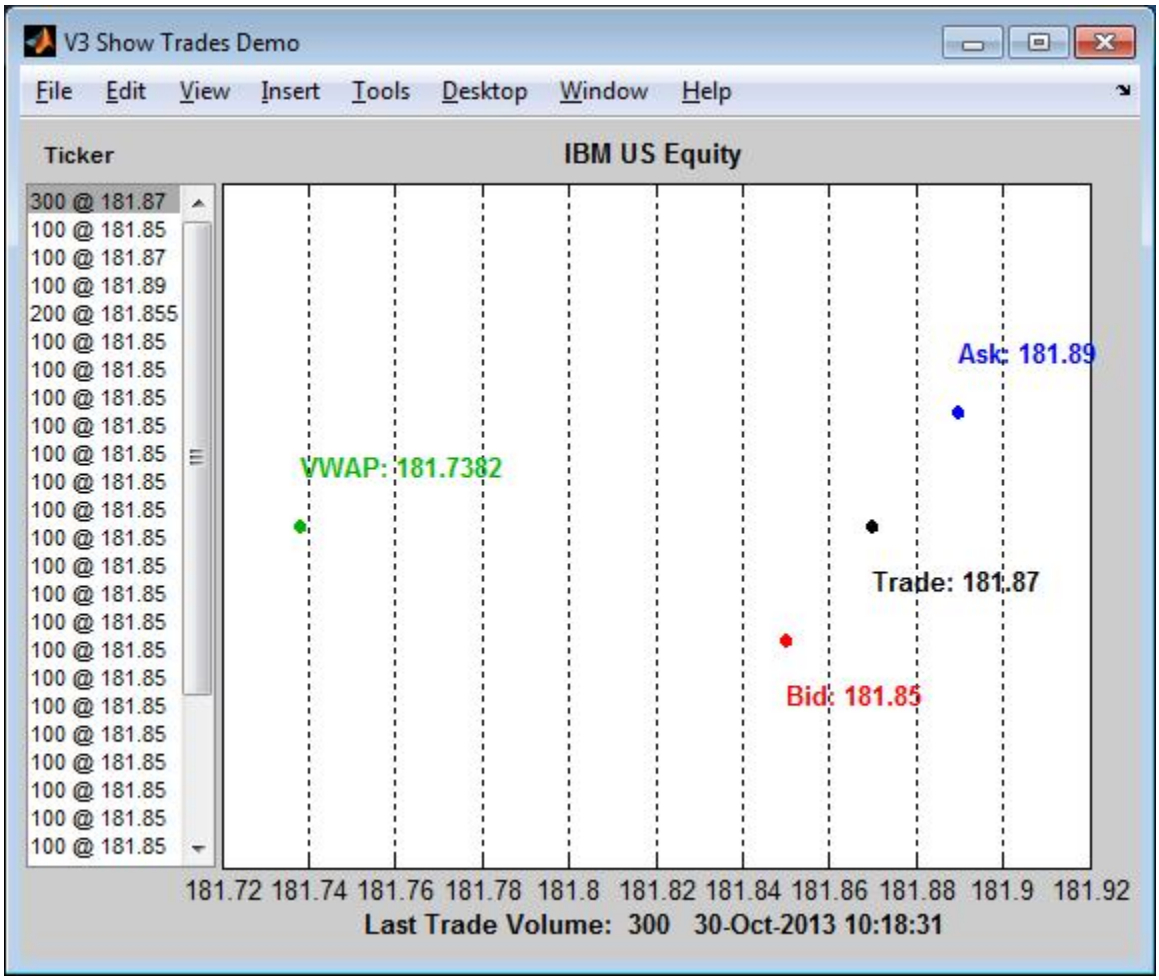
Retrieve volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for the IBM security using the event handler `v3showtrades`.

`v3showtrades` requires the input argument `f` of `realtime` to be any combination of: `'Last_Trade'`, `'Bid'`, `'Ask'`, `'Volume'`, and `'VWAP'`.

```
[subs,t] = realtime(c,'IBM US Equity',...
                    {'Last_Trade','Bid','Ask','Volume','VWAP'},...
                    'v3showtrades')
```

```
subs =

com.bloomberglp.blpapi.SubscriptionList@5c17dcdb


   Timer Object: timer-4

   Timer Settings
      ExecutionMode: fixedRate
             Period: 0.05
          BusyMode: drop
           Running: on

   Callbacks
          TimerFcn: 1x4 cell array
          ErrorFcn: ''
          StartFcn: ''
           StopFcn: ''
```

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `v3showtrades` displays a figure showing volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for IBM.




Real-time data continues to display until you execute the `stop` or `close` function.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Data for One Security Using the Event Handler `v3pricevol`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3pricevol` that creates a figure showing last price and volume data for a security.

Create the Bloomberg connection.

```
c = blp;
```

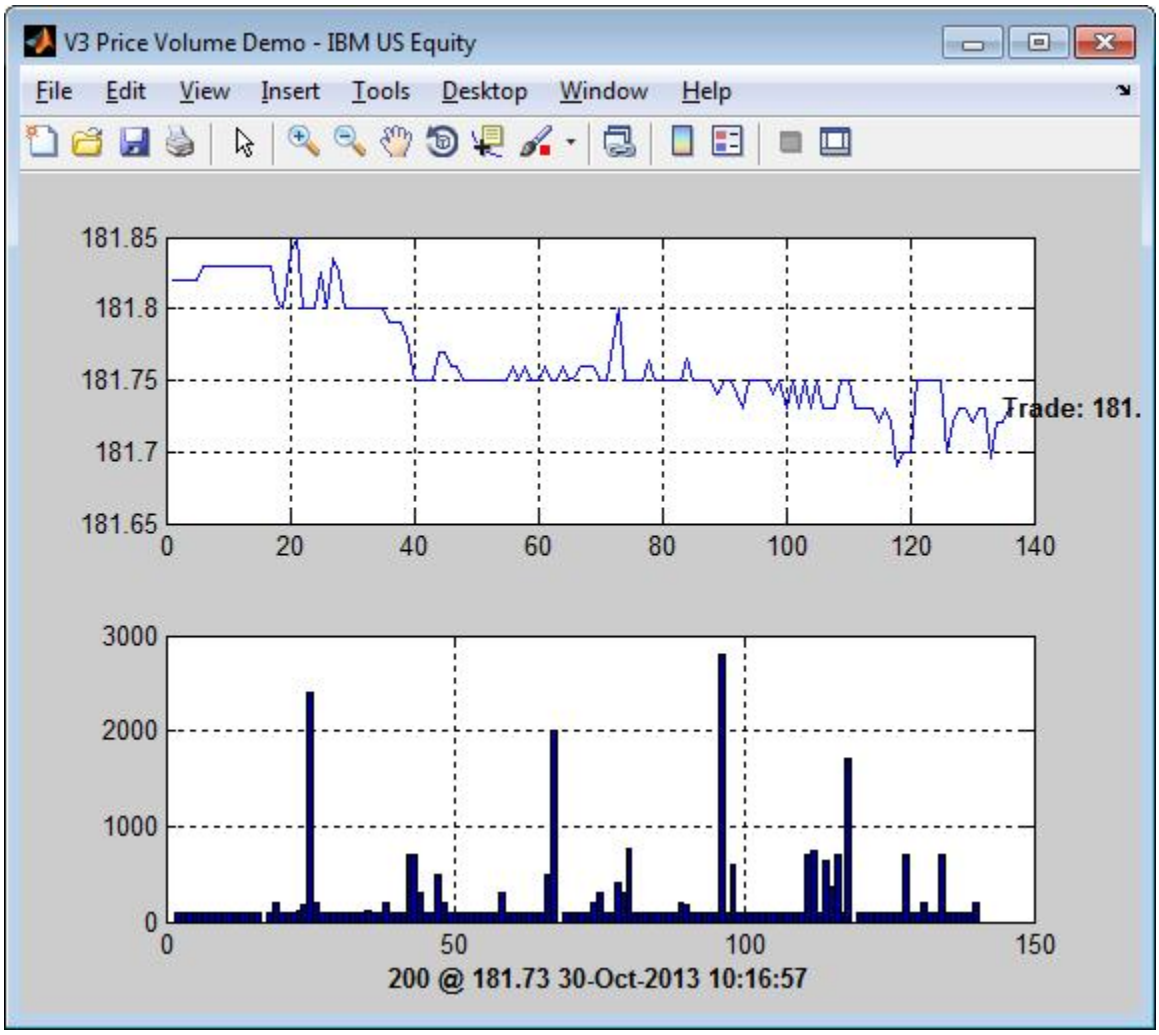
Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve last price and volume data for the IBM security using event handler `v3pricevol`.

`v3pricevol` requires the input argument `f` of `realtime` to be `'Last_Price'`, `'Volume'`, or both.

```
[subs,t] = realtime(c,'IBM US Equity',{'Last_Price','Volume'},...
                    'v3pricevol')

subs =

com.bloomberglp.blpapi.SubscriptionList@16f66676


   Timer Object: timer-5

   Timer Settings
      ExecutionMode: fixedRate
             Period: 0.05
          BusyMode: drop
           Running: on

   Callbacks
          TimerFcn: 1x4 cell array
          ErrorFcn: ''
          StartFcn: ''
           StopFcn: ''
```

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `v3pricevol` displays a figure showing last price and volume data for IBM.




Real-time data continues to display until you execute the `stop` or `close` function.

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### s — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector or string scalar for one security or a cell array of character vectors or string array for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell` | `string`

**`f` — Bloomberg data fields**
character vector | string scalar | cell array of character vectors | string array

Bloomberg data fields, specified as a character vector, string scalar, cell array of character vectors, or string array. A character vector or string denotes one Bloomberg data field name. A cell array of character vectors or string array denotes multiple Bloomberg data field names. For details about the fields you can specify, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Example: `{'LAST_PRICE';'OPEN'}`

Data Types: `char` | `cell` | `string`

**`eventhandler` — Event handler**
character vector | string scalar

Event handler, specified as a character vector or string scalar that denotes the name of an event handler function that you define. You can define an event handler function to process any type of real-time Bloomberg events. The specified event handler function runs every time the timer fires.

Data Types: `char` | `string`

## Output Arguments

**`d` — Bloomberg data**
structure (default) | table | timetable

Bloomberg data, returned as a structure, table, or timetable. The data type of the Bloomberg data depends on the DataReturnFormat and DatetimeType properties of the connection object. For details about the data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**`subs` — Bloomberg subscription**
object

Bloomberg subscription, returned as a Bloomberg object. For details about this object, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**`t` — MATLAB timer**
object

MATLAB timer, returned as a MATLAB object. For details about this object, see `timer`.

## See Also

`blp` | `close` | `getdata` | `history` | `stop` | `timeseries`

**Topics**
"Retrieve Bloomberg Real-Time Data" on page 3-14
"Workflow for Bloomberg" on page 3-19
"Writing and Running Custom Event Handler Functions" on page 1-26

**Introduced in R2010a**

# stop

Unsubscribe real-time requests for Bloomberg connection V3

## Syntax

```
stop(c,subs,t)
stop(c,subs,[],s)
```

## Description

`stop(c,subs,t)` unsubscribes real-time requests associated with the Bloomberg connection `c` and subscription list `subs`. `t` is the timer associated with the real-time callback for the subscription list.

`stop(c,subs,[],s)` unsubscribes real-time requests for each security `s` on the subscription list `subs`. The timer input is empty.

## Examples

### Stop Real-Time Requests

Unsubscribe to real-time data for one security.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume for the IBM security using the event handler `v3stockticker`.

`v3stockticker` requires the input argument `f` of `realtime` to be `'Last_Trade'`, `'Volume'`, or both.

```
[subs,t] = realtime(c,'IBM US Equity',{'Last_Trade','Volume'},...
                    'v3stockticker');
```

```
** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

`realtime` returns the stock tick data for the IBM security with the volume and last trade price.

Stop the real-time data requests for the IBM security using the Bloomberg subscription `subs` and MATLAB timer object `t`.

```
stop(c,subs,t)
```

Close the Bloomberg connection.

```
close(c)
```

### Stop Real-Time Requests for a Security List

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the last trade and volume for the security list `s` using the event handler `v3stockticker`. `s` contains securities for IBM, Google, and Ford Motor Company.

`v3stockticker` requires the input argument `f` of `realtime` to be `'Last_Trade'`, `'Volume'`, or both.

```
s = {'IBM US Equity','GOOG US Equity','F US Equity'};
[subs,t] = realtime(c,s,{'Last_Trade','Volume'},'v3stockticker');
```

```
** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

`realtime` returns the stock tick data for the securities list `s` with the volume and last trade price.

Stop the real-time data requests for the securities list `s` using the Bloomberg subscription `subs`.

```
stop(c,subs,[],s)
```

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `subs` — Bloomberg subscription

object

Bloomberg subscription, specified as a Bloomberg object. For details about this object, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

### `t` — MATLAB timer

object

MATLAB timer, specified as a MATLAB object. For details about this object, see `timer`.

### `s` — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector or string scalar for one security or a cell array of character vectors or string array for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell` | `string`

## See Also

`blp` | `close` | `getdata` | `history` | `realtime` | `timeseries`

**Topics**

“Retrieve Bloomberg Real-Time Data” on page 3-14
“Workflow for Bloomberg” on page 3-19

**Introduced in R2010a**

# tahistory

Historical technical analysis for Bloomberg connection V3

## Syntax

```
d = tahistory(c)
d = tahistory(c,s,startdate,enddate,study,period,Name,Value)
```

## Description

`d = tahistory(c)` returns the Bloomberg V3 session technical analysis data study and element definitions.

`d = tahistory(c,s,startdate,enddate,study,period,Name,Value)` returns the Bloomberg V3 session technical analysis data study and element definitions with additional options specified by one or more name-value pair arguments.

## Examples

### Request Bloomberg Directional Movement Indicator (DMI) Study for Security

Return all available Bloomberg studies and use the DMI study to run a technical analysis for a security.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

List the available Bloomberg studies.

```
d = tahistory(c)
```

```
d =

         dmiStudyAttributes: [1x1 struct]
       smavgStudyAttributes: [1x1 struct]
        bollStudyAttributes: [1x1 struct]
         maoStudyAttributes: [1x1 struct]
          fgStudyAttributes: [1x1 struct]
         rsiStudyAttributes: [1x1 struct]
        macdStudyAttributes: [1x1 struct]
         tasStudyAttributes: [1x1 struct]
       emavgStudyAttributes: [1x1 struct]
      maxminStudyAttributes: [1x1 struct]
        ptpsStudyAttributes: [1x1 struct]
        cmciStudyAttributes: [1x1 struct]
        wlprStudyAttributes: [1x1 struct]
       wmavgStudyAttributes: [1x1 struct]
```

```
     trenderStudyAttributes: [1x1 struct]
         gocStudyAttributes: [1x1 struct]
        kltnStudyAttributes: [1x1 struct]
    momentumStudyAttributes: [1x1 struct]
         rocStudyAttributes: [1x1 struct]
         maeStudyAttributes: [1x1 struct]
       hurstStudyAttributes: [1x1 struct]
        chkoStudyAttributes: [1x1 struct]
          teStudyAttributes: [1x1 struct]
       vmavgStudyAttributes: [1x1 struct]
       tmavgStudyAttributes: [1x1 struct]
         atrStudyAttributes: [1x1 struct]
         rexStudyAttributes: [1x1 struct]
         adoStudyAttributes: [1x1 struct]
          alStudyAttributes: [1x1 struct]
         etdStudyAttributes: [1x1 struct]
         vatStudyAttributes: [1x1 struct]
        tvatStudyAttributes: [1x1 struct]
          pdStudyAttributes: [1x1 struct]
          rvStudyAttributes: [1x1 struct]
      ipmavgStudyAttributes: [1x1 struct]
       pivotStudyAttributes: [1x1 struct]
          orStudyAttributes: [1x1 struct]
         pcrStudyAttributes: [1x1 struct]
          bsStudyAttributes: [1x1 struct]
```

`d` contains structures pertaining to each available Bloomberg study.

Display the name-value pairs for the DMI study.

```
d.dmiStudyAttributes
```

```
ans =

               period: [1x104 char]
     priceSourceHigh: [1x123 char]
      priceSourceLow: [1x121 char]
    priceSourceClose: [1x125 char]
```

Obtain more information about the `period` property.

```
d.dmiStudyAttributes.period
```

```
ans =

DEFINITION period {

    Min Value = 1

    Max Value = 1

    TYPE Int64

} // End Definition: period
```

Run the DMI study for the IBM security for the last month with `period` equal to `14`, the high price, the low price, and the closing price.

```
d = tahistory(c,'IBM US Equity',floor(now)-30,floor(now),'dmi',...
              'all_calendar_days','period',14,...
```

```
            'priceSourceHigh','PX_HIGH',...
            'priceSourceLow','PX_LOW','priceSourceClose','PX_LAST')
```

```
d =

         date: [31x1 double]
     DMI_PLUS: [31x1 double]
    DMI_MINUS: [31x1 double]
          ADX: [31x1 double]
         ADXR: [31x1 double]
```

`d` contains a `studyDataTable` with one `studyDataRow` for each interval returned.

Display the first five dates in the returned data.

```
d.date(1:5,1)
```

```
ans =

     735507.00
     735508.00
     735509.00
     735510.00
     735511.00
```

Display the first five prices in the plus DI line.

```
d.DMI_PLUS(1:5,1)
```

```
ans =

         18.92
         17.84
         16.83
         15.86
         15.63
```

Display the first five prices in the minus DI line.

```
d.DMI_MINUS(1:5,1)
```

```
ans =

         30.88
         29.12
         28.16
         30.67
         29.24
```

Display the first five values of the Average Directional Index.

```
d.ADX(1:5,1)
```

```
ans =

         22.15
         22.28
         22.49
         23.15
         23.67
```

Display the first five values of the Average Directional Movement Index Rating.

```
d.ADXR(1:5,1)
```

```
ans =

         25.20
         25.06
         25.05
         25.60
         26.30
```

Close the Bloomberg connection.

```
close(c)
```

### Request DMI Study for Security with Pricing Source

Run a technical analysis to return the DMI study for a security with a pricing source.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Run the DMI study for the Microsoft security with pricing source `ETPX` for the last month with `period` equal to `14`, the high price, the low price, and the closing price.

```
d = tahistory(c,'MSFT@ETPX US Equity',floor(now)-30,floor(now),...
              'dmi','all_calendar_days','period',14,...
              'priceSourceHigh','PX_HIGH','priceSourceLow','PX_LOW',...
              'priceSourceClose','PX_LAST')
```

```
d =

         date: [31x1 double]
     DMI_PLUS: [31x1 double]
    DMI_MINUS: [31x1 double]
          ADX: [31x1 double]
         ADXR: [31x1 double]
```

`d` contains a `studyDataTable` with one `studyDataRow` for each interval returned.

Display the first five dates in the returned data.

```
d.date(1:5,1)
```

```
ans =

     735507.00
     735508.00
     735509.00
     735510.00
     735511.00
```

Display the first five prices in the plus DI line.

```
d.DMI_PLUS(1:5,1)
```

```
ans =

         28.37
         30.63
         32.72
         30.65
         29.37
```

Display the first five prices in the minus DI line.

```
d.DMI_MINUS(1:5,1)
```

```
ans =

         21.97
         21.17
         19.47
         18.24
         17.48
```

Display the first values of the Average Directional Index.

```
d.ADX(1:5,1)
```

```
ans =

         13.53
         13.86
         14.69
         15.45
         16.16
```

Display the first five values of the Average Directional Movement Index Rating.

```
d.ADXR(1:5,1)
```

```
ans =

         15.45
         15.36
         15.53
         15.85
         16.37
```

Close the Bloomberg connection.

```
close(c)
```

### Return DMI Study Data as Table with Dates

Create a Bloomberg® connection, and then return data for a DMI study. The `tahistory` function returns data for dates as a `datetime` array.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `tahistory` function returns data as a structure.

Return dates as a `datetime` array by setting the `DatetimeType` property of the connection object. In this case, the table contains dates in variables that are `datetime` arrays.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Run the DMI study for the IBM® security from June 12, 2017 through June 16, 2017 with `period` equal to 14, the high price, the low price, and the closing price.

```
d = tahistory(c,'IBM US Equity','6/12/2017','6/16/2017','dmi', ...
    'all_calendar_days','period',14,'priceSourceHigh','PX_HIGH', ...
    'priceSourceLow','PX_LOW','priceSourceClose','PX_LAST');
```

Access the DMI study data for the first three dates.

```
d(1:3,:)
```

```
ans =

  3×5 table

       date        DMI_PLUS    DMI_MINUS     ADX     ADXR
    ___________    ________    _________    _____    _____

    12-Jun-2017     30.48        16.31      33.93    45.26
    13-Jun-2017     28.88        15.45      33.67    44.10
    14-Jun-2017     26.62        18.98      32.46    42.67
```

`d` is a `table` that contains these columns:

- `date` -- Date
- `DMI_PLUS` -- Prices in plus DI line
- `DMI_MINUS` -- Prices in minus DI line
- `ADX` -- Average Directional Index values
- `ADXR` -- Average Directional Movement Index Rating values

Access the first three dates in the returned data.

```
d.date(1:3)
```

```
ans =
```

```
  3×1 datetime array

   12-Jun-2017
   13-Jun-2017
   14-Jun-2017
```

Close the Bloomberg connection.

```
close(c)
```

### Return DMI Study Data as Timetable

Create a Bloomberg® connection, and then return data for a DMI study. The `tahistory` function returns data as a `timetable`.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `tahistory` function returns data as a structure.

```
c.DataReturnFormat = 'timetable';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Run the DMI study for the IBM® security from June 12, 2017 through June 16, 2017 with `period` equal to 14, the high price, the low price, and the closing price.

```
d = tahistory(c,'IBM US Equity','6/12/2017','6/16/2017','dmi', ...
    'all_calendar_days','period',14,'priceSourceHigh','PX_HIGH', ...
    'priceSourceLow','PX_LOW','priceSourceClose','PX_LAST');
```

Access the DMI study data for the first three dates.

```
d(1:3,:)
```

```
ans =

  3×4 timetable

      date        DMI_PLUS    DMI_MINUS     ADX     ADXR
   ___________    ________    _________    _____    _____

   12-Jun-2017     30.48        16.31      33.93    45.26
   13-Jun-2017     28.88        15.45      33.67    44.10
   14-Jun-2017     26.62        18.98      32.46    42.67
```

`d` is a `timetable` that contains these columns:

- `date` -- Date
- `DMI_PLUS` -- Prices in plus DI line
- `DMI_MINUS` -- Prices in minus DI line
- `ADX` -- Average Directional Index values
- `ADXR` -- Average Directional Movement Index Rating values

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### `c` — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### `s` — Security

character vector | string scalar

Security, specified as a character vector or string scalar for a single Bloomberg security.

Data Types: `char` | `string`

### `startdate` — Start date

numeric scalar | character vector | string scalar

Start date, specified as a numeric scalar, character vector, or string scalar to denote the start date of the date range for the returned tick data.

Example: `floor(now-1)`

Data Types: `double` | `char` | `string`

### `enddate` — End date

numeric scalar | character vector | string scalar

End date, specified as a numeric scalar, character vector, or string scalar to denote the end date of the date range for the returned tick data.

Example: `floor(now)`

Data Types: `double` | `char` | `string`

### `study` — Study type

character vector | string scalar

Study type, specified as a character vector or string scalar to denote the study to use for historical analysis.

Data Types: `char` | `string`

### `period` — Periodicity

`'daily'` | `'weekly'` | `'monthly'` | `'quarterly'` | ...

Periodicity, specified as one of these values to denote the data to return. For specifying multiple values, use a cell array. For example, when `period` is set to `{'daily','all_calendar_days'}`,

`tahistory` returns daily data for all calendar days, and reports missing data as `NaN`s. When `period` is set to `'active_days_only'`, `tahistory` returns data using the default periodicity for active trading days only. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. These tables show the values for `period`.

To specify the periodicity of the return data, see this table.

| Value | Description |
|---|---|
| `'daily'` | Return data for each day. |
| `'weekly'` | Return data for each week. |
| `'monthly'` | Return data for each month. |
| `'quarterly'` | Return data for each quarter. |
| `'semi_annually'` | Return data semiannually. |
| `'yearly'` | Return data for each year. |

The anchor date is the date to which all other reported dates are related. To specify the anchor date, see this table.

| Value | Description |
|---|---|
| `'actual'` | Anchor date specification for an actual date. For this function, for periodicities other than daily, `enddate` is the anchor date.<br><br>If the period is weekly and the `enddate` is a Thursday, every data point is a Thursday, or the nearest prior business day to Thursday. If the period is monthly and the `enddate` is the 20th of a month, every data point is the 20th of each month in the date range. |
| `'calendar'` | Anchor date specification for a calendar year. |
| `'fiscal'` | Anchor date specification for a fiscal year. |

To specify returning data for particular days, see this table.

| Value | Description |
|---|---|
| `'non_trading_weekdays'` | Return data for all weekdays. |
| `'all_calendar_days'` | Return data for all calendar days. |
| `'active_days_only'` | Return data for only active trading days. |

To specify how to fill missing values, see this table.

| Value | Description |
|---|---|
| `'previous_value'` | Fill missing values with previous values for dates without trading activity for the security. |
| `'nil_value'` | Fill missing values with a `NaN` for dates without trading activity for the security. |

Data Types: `char` | `cell`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'period',14,'priceSourceHigh','PX_HIGH','priceSourceLow','PX_LOW','priceSourceClose','PX_LAST'`

**Note** For details about the full list of name-value pair arguments, see the Bloomberg tool located at `C:\blp\API\APIv3\bin\BBAPIDemo.exe`.

#### `period` — Period
numeric scalar

Period, specified as the comma-separated pair consisting of `'period'` and a numeric scalar. For details about the period, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `double`

#### `priceSourceHigh` — High price
character vector | string scalar

High price, specified as the comma-separated pair consisting of `'priceSourceHigh'` and a character vector or string scalar. For details about the high price, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `char` | `string`

#### `priceSourceLow` — Low price
character vector | string scalar

Low price, specified as the comma-separated pair consisting of `'priceSourceLow'` and a character vector or string scalar. For details about the low price, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `char` | `string`

#### `priceSourceClose` — Closing price
character vector | string scalar

Closing price, specified as the comma-separated pair consisting of `'priceSourceClose'` and a character vector or string scalar. For details about the closing price, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `char` | `string`

## Output Arguments

#### `d` — Technical analysis data
structure (default) | table | timetable

Technical analysis data, returned as a structure, table, or timetable. The data type of the returned data depends on the DataReturnFormat and DatetimeType properties of the connection object.

For details about the data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

## See Also

`blp` | `close` | `getdata` | `history` | `realtime` | `timeseries`

### Topics

"Retrieve Bloomberg Current Data" on page 3-6
"Retrieve Current and Historical Data Using Bloomberg" on page 1-14
"Workflow for Bloomberg" on page 3-19

### Introduced in R2012b

# timeseries

Intraday tick data for Bloomberg connection V3

## Syntax

```
d = timeseries(c,s,date)
d = timeseries(c,s,date,interval,field)
d = timeseries(c,s,date,[],field,options,values)

d = timeseries(c,s,{startdate,enddate})
d = timeseries(c,s,{startdate,enddate},interval,field)
d = timeseries(c,s,{startdate,enddate},[],field)
d = timeseries(c,s,{startdate,enddate},[],field,options,values)

d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval)
d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval,field)

d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime},
interval)
d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime},
interval,field)
```

## Description

`d = timeseries(c,s,date)` retrieves raw tick data using the connection object and a security for a specific date.

`d = timeseries(c,s,date,interval,field)` retrieves raw tick data that is aggregated into intervals for a specific field.

`d = timeseries(c,s,date,[],field,options,values)` retrieves raw tick data without an aggregation interval for a specific field with the specified options and corresponding values.

`d = timeseries(c,s,{startdate,enddate})` retrieves raw tick data for a date range using a start date and an end date.

`d = timeseries(c,s,{startdate,enddate},interval,field)` retrieves raw tick data for a specific date range aggregated into intervals for a specific field.

`d = timeseries(c,s,{startdate,enddate},[],field)` retrieves raw tick data for a specific date range without an aggregation interval for a specific field.

`d = timeseries(c,s,{startdate,enddate},[],field,options,values)` retrieves raw tick data for a specific date range without an aggregation interval for a specific field with specified options and corresponding values.

`d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval)` retrieves raw trade tick data for a specific time range for each day within a specific date range, aggregated into intervals.

`d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval,field)` uses a specific field for tick data to return.

`d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime}, interval)` retrieves raw trade tick data for a whole day increment within a specific date and time range, aggregated into intervals.

`d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime}, interval,field)` uses a specific field for tick data to return.

## Examples

### Retrieve Tick Data for Specific Date and Pricing Source

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date. Use a security with and without a pricing source to retrieve tick data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the trade tick series using the IBM security for today.

```
d = timeseries(c,'IBM US Equity',floor(now))
```

```
d =

    'TRADE'    [735537.40]    [181.69]    [100.00]
    'TRADE'    [735537.40]    [181.69]    [100.00]
    'TRADE'    [735537.40]    [181.68]    [100.00]
    ...
```

The columns in `d` are:

- Tick type
- Numeric representation of the date and time
- Tick value
- Tick size

Here, the first row shows that 100 IBM shares sold for $181.69 today.

Retrieve the trade tick series using the Microsoft security with pricing source ETPX for today.

```
d = timeseries(c,'MSFT@ETPX US Equity',floor(now))
```

```
d =

    'TRADE'    [735537.40]    [35.53]    [100.00]
    'TRADE'    [735537.40]    [35.55]    [200.00]
    'TRADE'    [735537.40]    [35.55]    [100.00]
    ...
```

Here, the first row shows that 100 Microsoft shares are sold for $35.53 today.

Close the Bloomberg connection.

```
close(c)
```

### Time Interval with Specific Field

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date. Specify the tick data to return using a time interval and field.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the trade tick series using the IBM security aggregated into 5-minute intervals for today.

```
d = timeseries(c,'IBM US Equity',floor(now),5,'Trade')
```

```
d =

  Columns 1 through 7

   735537.40        181.69        181.99        180.10        181.84     252322.00        861.00
   735537.40        181.90        181.97        181.57        181.65      78570.00        535.00
   735537.40        181.73        182.18        181.58        182.07     124898.00        817.00
     ...

  Column 8

   45815588.00
   14282076.00
   22710954.00
   ...
```

The columns in `d` contain the following:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

Here, the first row of data shows prices and tick data for the current date. The next row shows tick data for 5 minutes later.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Tick Data Using Option and Value

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date and field. Use option and value to return additional data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the trade tick series using the `'F US Equity'` security without specifying the aggregation parameter for today. Also, return the condition codes.

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
               'includeConditionCodes','true')
```

```
d =

    'TRADE'    [735556.57]    [17.12]    [   100.00]    'R6,IS'
    'TRADE'    [735556.57]    [17.12]    [   100.00]    ''
    'TRADE'    [735556.57]    [17.12]    [   500.00]    ''
    ...
```

The columns in `d` contain the following:

- Tick type
- Numeric representation of the date and time
- Tick value
- Tick size
- Condition codes

Here, the first row shows that 100 `'F US Equity'` security shares sold for $17.12 today.

Close the Bloomberg connection.

```
close(c)
```

### Retrieve Tick Data Using Date Range

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date range.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the tick series for the `'F US Equity'` security for the last business day from the beginning of the day to noon.

```
d = timeseries(c,'F US Equity',{floor(now-4),floor(now-3.5)})
```

```
d =

    'TRADE'    [735552.67]    [17.09]    [   200.00]
    'TRADE'    [735552.67]    [17.09]    [   100.00]
    'TRADE'    [735552.67]    [17.09]    [   100.00]
   ...
```

The columns in `d` are:

- Tick type
- Numeric representation of the date and time
- Tick value
- Tick size

Here, the first row shows that 200 `'F US Equity'` security shares were sold for $17.09 on the last business day.

Close the Bloomberg connection.

```
close(c)
```

### Date Range with Interval and Specific Field

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date range. Specify the interval and field.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Retrieve the trade tick series for the past 50 days for the IBM security aggregated into 5-minute intervals.

```
d = timeseries(c,'IBM US Equity',{floor(now)-50,floor(now)},5,'Trade')
```

```
ans =

  Columns 1 through 7

   735487.40        187.20        187.60        187.02        187.08     207683.00        560.00
   735487.40        187.03        187.13        186.65        186.78      46990.00        349.00
   735487.40        186.78        186.78        186.40        186.47      51589.00        399.00
     ...

Column 8

   38902968.00
    8779374.00
    9626896.00
   ...
```

The columns in `d` contain the following:

- Numeric representation of date and time
- Open price

- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row of data shows prices and tick data for the current date. The next row shows tick data for 5 minutes later.

Close the Bloomberg connection.

```
close(c)
```

### Date Range with Numerous Fields

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date range and numerous fields.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the Bid, Ask, and trade tick series for the security `'F US Equity'` for yesterday with a time interval at noon, without specifying the aggregation parameter.

```
d = timeseries(c,'F US Equity',{floor(now-1)+.5,floor(now-1)+.51},...
               [],{'Bid','Ask','Trade'})
```

```
d =

    'TRADE'    [735550.50]    [16.71]    [100.00]
    'ASK'      [735550.50]    [16.71]    [312.00]
    'BID'      [735550.50]    [16.70]    [177.00]
    ...
```

The columns in `d` are:

- Tick type
- Numeric representation of the date and time
- Tick value
- Tick size

Here, the first row shows that 100 `'F US Equity'` security shares sold for $16.71 yesterday.

Close the Bloomberg connection.

```
close(c)
```

### Date Range with Options and Values

First, create a Bloomberg Desktop connection. Then, retrieve tick data for a specific date range. Specify options and values to return additional data.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Return the trade tick series for the security `'F US Equity'` for yesterday with a time interval at noon, without specifying the aggregation parameter. Also, return the condition codes, exchange codes, and broker codes.

```
d = timeseries(c,'F US Equity',{floor(now-1)+.5,floor(now-1)+.51},...
               [],'Trade',{'includeConditionCodes',...
               'includeExchangeCodes','includeBrokerCodes'},...
               {'true','true','true'})
```

```
d =

    'TRADE'    [735550.50]    [16.71]    [100.00]    'T'     'D'
    'TRADE'    [735550.50]    [16.70]    [400.00]    'IS'    'B'
    'TRADE'    [735550.50]    [16.70]    [100.00]    'IS'    'B'
    ...
```

The columns in `d` contain the following:

- Tick type
- Numeric representation of the date and time
- Tick value
- Tick size
- Exchange condition codes
- Exchange codes

Broker codes are available for Canadian, Finnish, Mexican, Philippine, and Swedish equities only. In this case, the broker buy code appears in the seventh column and the broker sell code appears in the eighth column.

Here, the first row shows that 100 `'F US Equity'` security shares sold for $16.71 yesterday.

Close the Bloomberg connection.

```
close(c)
```

### Date and Time Range with Interval

Use Bloomberg® to retrieve raw trade tick data by specifying a time range for each day in a specific date range. Specify the time interval for the tick data.

Create the Bloomberg® connection.

```
c = blp;
```

Alternatively, you can connect to Bloomberg® Server using `blpsrv` or Bloomberg® B-PIPE® using `bpipe`.

Retrieve the trade tick series for the `'F US Equity'` security for the last two days. Use the time range from the beginning of the trading day through noon. Retrieve tick data aggregated into 5-minute intervals. `d` is a numeric matrix.

```
s = 'F US Equity';
startdate = datetime('today')-1;
enddate = datetime('today');
starttime = "09:30:00";
endtime = "12:00:00";
interval = 5;

d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval);
```

Set the display output for currency.

```
format bank
```

Display the first three ticks.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    736959.40         11.71         11.81         11.71         11.79
    736959.40         11.79         11.81         11.75         11.79
    736959.40         11.80         11.82         11.78         11.80

  Columns 6 through 8

   1375547.00       1190.00   16196757.00
    598924.00        898.00    7058724.00
    488655.00        641.00    5768371.50
```

The columns in `d` are:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row shows tick data at the start time of the time range. The next row shows tick data for 5 minutes later.

Determine the maximum high price for the last two days.

```
highprices = d(:,3);
m = max(highprices)
```

```
m =

         11.82
```

Close the Bloomberg® connection.

```
close(c)
```

### Date and Time Range with Interval and Specific Field

Use Bloomberg® to retrieve raw tick data by specifying a time range for each day in a specific date range. Specify the time interval and the field for the type of tick data to return. Here, specify the bid tick data.

Create the Bloomberg® connection.

```
c = blp;
```

Alternatively, you can connect to Bloomberg® Server using `blpsrv` or Bloomberg® B-PIPE® using `bpipe`.

Retrieve the tick series for the `'F US Equity'` security for the last two days. Use the time range from the beginning of the trading day through noon. Retrieve tick data aggregated into 5-minute intervals. Specify retrieving the bid tick series. `d` is a numeric matrix.

```
s = 'F US Equity';
startdate = datetime('today')-1;
enddate = datetime('today');
starttime = "09:30:00";
endtime = "12:00:00";
interval = 5;
field = 'BID';

d = timeseries(c,s,{startdate:enddate,starttime,endtime},interval,field);
```

Set the display output for currency.

```
format bank
```

Display the first three ticks.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    736959.40         11.70         11.80         11.70         11.79
```

```
    736959.40         11.79         11.80         11.75         11.79
    736959.40         11.79         11.81         11.78         11.80

  Columns 6 through 8

    397711.00       1442.00    4681704.50
    450997.00       1698.00    5311330.50
    464761.00       1391.00    5481707.50
```

The columns in `d` are:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row shows tick data at the start time of the time range. The next row shows tick data for 5 minutes later.

Determine the maximum high price for the last two days.

```
highprices = d(:,3);
m = max(highprices)
```

```
m =

         11.81
```

Close the Bloomberg® connection.

```
close(c)
```

### Date and Time Range with Day Increment and Interval

Use Bloomberg® to retrieve raw trade tick data by specifying a time range for each day in a specific date range. Specify a day increment for the date range and the time interval for the tick data.

Create the Bloomberg® connection.

```
c = blp;
```

Alternatively, you can connect to Bloomberg® Server using `blpsrv` or Bloomberg® B-PIPE® using `bpipe`.

Retrieve the trade tick series for the `'IBM US Equity'` security for the last two months. Set the day increment to 5 days. Use the time range from the beginning of the trading day through noon. Retrieve tick data aggregated into 5-minute intervals. `d` is a numeric matrix.

```
s = 'IBM US Equity';
startdate = datetime('today')-60;
enddate = datetime('today');
dayincrement = 5;
starttime = "09:30:00";
endtime = "12:00:00";
interval = 5;

d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime}, ...
    interval);
```

Set the display output for currency.

```
format bank
```

Display the first three ticks.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    736900.40        147.00        147.04        146.55        146.62
    736900.40        146.62        146.87        146.62        146.71
    736900.40        146.72        146.79        146.52        146.54

  Columns 6 through 8

    125558.00        393.00   18440146.00
     39535.00        258.00    5800969.00
     49659.00        314.00    7282961.00
```

The columns in `d` are:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row shows tick data at the start time of the time range. The next row shows tick data for 5 minutes later.

After the tick data for the first day in the date range, `d` contains tick data for a trading day that is 5 days later.

Close the Bloomberg® connection.

```
close(c)
```

### Date and Time Range with Day Increment, Interval, and Specific Field

Use Bloomberg® to retrieve raw tick data by specifying a time range for each day in a specific date range. Specify a day increment for the date range, the time interval for the tick data, and the field for the type of tick data to return. Here, specify the bid tick data.

Create the Bloomberg® connection.

```
c = blp;
```

Alternatively, you can connect to Bloomberg® Server using `blpsrv` or Bloomberg® B-PIPE® using `bpipe`.

Retrieve the trade tick series for the `'F US Equity'` security for the last two months. Set the day increment to 5 days. Use the time range from the beginning of the trading day through noon. Retrieve tick data aggregated into 5-minute intervals. Specify the bid tick series. `d` is a numeric matrix.

```
s = 'F US Equity';
startdate = datetime('today')-60;
enddate = datetime('today');
dayincrement = 5;
starttime = "09:30:00";
endtime = "12:00:00";
interval = 5;
field = 'BID';

d = timeseries(c,s,{startdate:dayincrement:enddate,starttime,endtime}, ...
    interval,field);
```

Set the display output for currency.

```
format bank
```

Display the first three ticks.

```
d(1:3,:)
```

```
ans =

  Columns 1 through 5

    736900.40         11.50         11.54         11.49         11.50
    736900.40         11.50         11.50         11.48         11.48
    736900.40         11.48         11.49         11.44         11.44

  Columns 6 through 8

    422305.00       1158.00    4863894.00
    575966.00       1180.00    6617854.00
    288147.00       1489.00    3305491.75
```

The columns in `d` are:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

The first row shows tick data at the start time of the time range. The next row shows tick data for 5 minutes later.

After the tick data for the first day in the date range, `d` contains tick data for a trading day that is 5 days later.

Close the Bloomberg® connection.

```
close(c)
```

### Return Tick Data as Table with Dates

Create a Bloomberg® connection, and then return intraday tick data. The `timeseries` function returns data for dates as a `datetime` array.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a table by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `timeseries` function returns data as a numeric array.

Return dates as a `datetime` array by setting the `DatetimeType` property of the connection object. In this case, the table contains dates in variables that are `datetime` arrays.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Retrieve the trade tick series for the IBM® security aggregated into 5-minute intervals for today. `d` is a table that contains the tick series data.

```
s = 'IBM US Equity';
date = floor(now);
interval = 5;
```

```
field = 'Trade';

d = timeseries(c,s,date,interval,field);
```

Access the first three ticks of data.

```
d(1:3,:)
```

```
ans =

  3×8 table

      DATE         OPEN      HIGH      LOW      CLOSE      VOLUME      NUMBER_OF_TICKS    TOTAL_
    ___________    ______    ______    ______    ______    _________    _______________    ______

    21-Dec-2017    153.17    153.31    153.08    153.31    152524.00        442.00         233676
    21-Dec-2017    153.35    153.35    152.82    152.84     46051.00        291.00          70486
    21-Dec-2017    152.84    153.21    152.82    153.16     30966.00        225.00          47373
```

`d` contains columns with the following data:

- Date
- Open price
- High price
- Low price
- Closing price
- Volume
- Number of ticks
- Total tick value in the bar

Access the first three dates in the `DATE` column.

```
d.DATE(1:3)
```

```
ans =

  3×1 datetime array

   21-Dec-2017
   21-Dec-2017
   21-Dec-2017
```

Close the Bloomberg connection.

```
close(c)
```

### Return Tick Data as Timetable

Create a Bloomberg® connection, and then return intraday tick data. The `timeseries` function returns data for dates as a `timetable`.

Create the Bloomberg connection.

```
c = blp;
```

Alternatively, you can connect to the Bloomberg Server using `blpsrv` or Bloomberg B-PIPE® using `bpipe`.

Return data as a `timetable` by setting the `DataReturnFormat` property of the connection object. If you do not set this property, the `timeseries` function returns data as a numeric array.

```
c.DataReturnFormat = 'timetable';
```

Adjust the display format of the returned data for currency.

```
format bank
```

Retrieve the trade tick series for the IBM® security aggregated into 5-minute intervals for today. `d` is a `timetable` that contains the tick series data.

```
s = 'IBM US Equity';
date = floor(now);
interval = 5;
field = 'Trade';

d = timeseries(c,s,date,interval,field);
```

Access the first three ticks of data.

```
d(1:3,:)
```

```
ans =

  3×7 timetable

      DATE         OPEN      HIGH      LOW      CLOSE      VOLUME      NUMBER_OF_TICKS    TOTAL_
    ___________    ______    ______    ______    ______    _________    _______________    ______

    21-Dec-2017    153.17    153.31    153.08    153.31    152524.00        442.00         23367
    21-Dec-2017    153.35    153.35    152.82    152.84     46051.00        291.00          7048
    21-Dec-2017    152.84    153.21    152.82    153.16     30966.00        225.00          4737
```

`d` is a `timetable` that contains the following data:

- Date
- Open price
- High price
- Low price
- Closing price
- Volume
- Number of ticks
- Total tick value in the bar

Close the Bloomberg connection.

```
close(c)
```

## Input Arguments

### c — Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`, `blpsrv`, or `bpipe`.

### s — Security

character vector | string scalar

Security, specified as a character vector or string scalar for a single Bloomberg security.

Data Types: `char` | `string`

### date — Date

numeric scalar | character vector | string scalar | `datetime` array

Date, specified as a numeric scalar, character vector, string scalar, or `datetime` array. `date` specifies the date for the returned tick data based on the entire day from midnight until 11:59:59 p.m.

Example: `floor(now)`

Data Types: `double` | `char` | `string` | `datetime`

### interval — Time interval

numeric scalar

Time interval, specified as a numeric scalar to denote the number of minutes between ticks for the returned tick data.

Data Types: `double`

### field — Bloomberg field

`'TRADE'` (default) | `'BID'` | `'ASK'` | ...

Bloomberg field, specified as one of these values that define the tick data to return.

| Request Type | Valid Bloomberg Field Values |
|---|---|
| IntradayBarRequest with time interval specified | `'TRADE'` |
| | `'BID'` |
| | `'ASK'` |
| | `'BID_BEST'` |
| | `'ASK_BEST'` |
| IntradayTickRequest with no time interval specified | `'TRADE'` |
| | `'BID'` |
| | `'ASK'` |
| | `'BID_BEST'` |
| | `'ASK_BEST'` |
| | `'SETTLE'` |

### `options` — Bloomberg API options

`'includeConditionCodes'` | `'includeExchangeCodes'` | `'includeBrokerCodes'` | ...

Bloomberg API options, specified as one of the values in this table.

| Value | Description |
|---|---|
| `'includeConditionCodes'` | Exchange condition codes associated with the event |
| `'includeExchangeCodes'` | Exchange code where tick originated |
| `'includeBrokerCodes'` | Broker code |
| `'includeRpsCodes'` | Reporting party side |
| `'includeNonPlottableEvents'` | After-hours data |

**Note** The value `'includeNonPlottableEvents'` applies to raw intraday requests only.

To specify more than one Bloomberg API option, use a cell array of these values.

Specify the corresponding Bloomberg API value for each API option. The number of options must match the number of values.

For example, to specify one Bloomberg API option, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
               'includeConditionCodes','true');
```

To specify two Bloomberg API options, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
               {'includeConditionCodes','includeExchangeCodes'},...
               {'true','true'});
```

For details about the options, see the *Bloomberg API Developer's Guide*.

Data Types: `char` | `cell`

### `values` — Bloomberg API values

`'true'` | `'false'`

Bloomberg API values, specified as `'true'` or `'false'`. Each value corresponds to the specified Bloomberg API option. To specify more than one Bloomberg API value, use a cell array. The number of values must match the number of options.

For example, to specify one Bloomberg API option, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
               'includeConditionCodes','true');
```

To specify two Bloomberg API options, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
               {'includeConditionCodes','includeExchangeCodes'},...
               {'true','true'});
```

Data Types: `char` | `cell`

**`startdate` — Start date**
numeric scalar | character vector | string scalar | `datetime` array

Start date, specified as a numeric scalar, character vector, string scalar, or `datetime` array. This date specifies the beginning of the date range for the returned tick data. If no ticks are present in the date range, then returned tick data is empty.

Example: `floor(now-1)`

Data Types: `double` | `char` | `string` | `datetime`

**`enddate` — End date**
numeric scalar | character vector | string scalar | `datetime` array

End date, specified as a numeric scalar, character vector, string scalar, or `datetime` array. This date specifies the end of the date range for the returned tick data. If no ticks are present in the date range, then returned tick data is empty.

Example: `floor(now)`

Data Types: `double` | `char` | `string` | `datetime`

**`starttime` — Start time**
character vector | string scalar | `datetime` array

Start time, specified as a character vector, string scalar, or `datetime` array. This time specifies the start time of the time range for the returned tick data.

Example: `'09:30:00'`

Data Types: `char` | `string` | `datetime`

**`endtime` — End time**
character vector | string scalar | `datetime` array

End time, specified as a character vector, string scalar, or `datetime` array. This time specifies the end time of the time range for the returned tick data.

Example: `'16:30:00'`

Data Types: `char` | `string` | `datetime`

**`dayincrement` — Day increment**
1 (default) | numeric scalar

Day increment, specified as a numeric scalar. This number specifies the whole day increment for a specific date range. For example, if the day increment is 7, then the returned data contains ticks for every 7th day starting from the first day within the date range.

Data Types: `double`

## Output Arguments

**`d` — Bloomberg tick data**
cell array | numeric array | table | timetable

Bloomberg tick data, returned as one of these data types:

- Cell array for requests without a specified time interval (raw tick data)
- Numeric array for requests with a specified time interval
- table
- timetable

The data type of the tick data depends on the DataReturnFormat and DatetimeType properties of the connection object.

**Note** The Bloomberg API returns the tick time with precision in seconds.

## Limitations

When the data request is too large, `timeseries` displays this error message:

```
Timeout error:
Error using blp/timeseries>processResponseEvent (line 338) REQUEST FAILED: responseError = {

source = bbdbl7

code = -2

category = TIMEOUT

message = Timed out getting data from store [nid:327]

subcategory = INTERNAL_ERROR

}
```

To fix this error, shorten the length of the date range by modifying the input arguments `startdate` and `enddate`.

## Tips

- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/javaclasspath.txt`. For details about the static Java class path, see “Static Path”.
- You cannot retrieve Bloomberg intraday tick data for a date more than 140 days ago.
- The *Bloomberg API Developer’s Guide* states that `'TRADE'` corresponds to LAST_PRICE for IntradayTickRequest and IntradayBarRequest.
- Bloomberg V3 intraday tick data supports additional name-value pairs. For details on these pairs, see the *Bloomberg API Developer’s Guide* by typing `WAPI` and clicking the **<GO>** button on the Bloomberg terminal.
- You can check data and field availability by using the Bloomberg Excel Add-In.

## See Also

`blp` | `close` | `getdata` | `history` | `realtime`

### Topics

“Retrieve Bloomberg Intraday Tick Data” on page 3-12
“Workflow for Bloomberg” on page 3-19

### Introduced in R2010a

# datastream

(To be removed) Thomson ReutersDatastream DataWorks API connection

**Note** The `datastream` object will be removed in a future release. Use the `datastreamws` object instead. For details, see “Compatibility Considerations”.

## Description

The `datastream` function creates a `datastream` object. The `datastream` object represents a Thomson Reuters® Datastream DataWorks® API connection.

After you create a `datastream` object, you can use the object functions to retrieve historical data for different securities, and retrieve fields for a specific date or date range.

## Creation

### Syntax

```
c = datastream(username,password,source)
c = datastream(username,password,source,url)
```

#### Description

`c = datastream(username,password,source)` creates a connection to the Thomson Reuters Datastream DataWorks API using a user name, password, and data source.

`c = datastream(username,password,source,url)` connects using a Thomson Reuters URL.

#### Input Arguments

**`username` — Thomson Reuters user name**
character vector | string scalar

Thomson Reuters user name, specified as a character vector or string scalar. To find your user name, contact Thomson Reuters.

**Note** The character vector or string scalar must start with `DS:` followed by the user name.

Example: `'DS:USER1'`

Data Types: `char` | `string`

**`password` — Thomson Reuters password**
character vector | string scalar

Thomson Reuters password, specified as a character vector or string scalar. To find your password, contact Thomson Reuters.

Example: `'XXXXXXX'`

Data Types: `char` | `string`

**`source` — Data source**
`'Datastream'` | `"Datastream"`

Data source, specified as the value `'Datastream'` or `"Datastream"`.

**`url` — Thomson Reuters URL**
character vector | string scalar

Thomson Reuters URL, specified as a character vector or string scalar. For details, contact Thomson Reuters.

Example: `'http://dataworks.thomson.com/Dataworks/Enterprise/1.0'`

Data Types: `char` | `string`

## Properties

**`user` — Thomson Reuters user name**
character vector

Thomson Reuters user name, specified as a character vector.

The `datastream` function sets this property using the `username` input argument.

Example: `'DS:USER1'`

Data Types: `char`

**`password` — Thomson Reuters password**
character vector

Thomson Reuters password, specified as a character vector. To find your password, contact Thomson Reuters.

The `datastream` function sets this property using the `password` input argument.

Example: `'XXXXXXX'`

Data Types: `char`

**`datasource` — Data source**
`'Datastream'`

Data source, specified as the value `'Datastream'`.

The `datastream` function sets this property using the `source` input argument.

**`endpoint` — API endpoint URL**
character vector

API endpoint URL, specified as a character vector.

The `datastream` function sets this property using the `url` input argument.

Example: `'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx'`

Data Types: `char`

**`wsdl` — API endpoint WSDL**
character vector

API endpoint WSDL, specified as a character vector. This property describes the endpoint for the messages associated with the Datastream DataWorks service.

Example: `'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx?WSDL'`

Data Types: `char`

## Object Functions

| | |
|---|---|
| fetch | (To be removed) Request data from Thomson Reuters Datastream data servers |
| get | (To be removed) Retrieve properties of Thomson Reuters Datastream connection objects |
| isconnection | (To be removed) Determine if connections to Thomson Reuters Datastream data servers are valid |
| close | (To be removed) Close connections to Thomson Reuters Datastream data servers |

## Examples

### Connect to Thomson Reuters Datastream

Create a Thomson Reuters Datastream connection. Then, retrieve closing and lowest prices for a security. The data you see when completing this example can differ from the output data shown.

Connect to Thomson Reuters Datastream using a user name, password, and data source. `c` is the Thomson Reuters Datastream connection object.

```
username = 'DS:USER1';
password = 'XXXXXXX';
source = 'Datastream';
c = datastream(username,password,source)
```

```
c =

  datastream with properties:

          user: 'DS:USER1''
      password: 'XXXXXXX'
    datasource: 'Datastream'
      endpoint: 'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx'
          wsdl: 'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx?WSDL
```

Using the IBM security, retrieve closing and lowest prices for yesterday. Each price value is returned as a field in the structure `d`.

```
sec = 'U:IBM';
prices = {'P','PL'};
yesterday = floor(now)-1;
```

```
d = fetch(c,sec,prices,yesterday);
```

Convert closing and lowest prices from character vectors to doubles, and display them using the `str2num` function.

```
closing = str2num(d.P);
lowest = str2num(d.PL);

[closing lowest]
```

```
ans =

        166.73        166.06
```

Close the Thomson Reuters Datastream connection.

```
close(c)
```

### Connect to Thomson Reuters Datastream Using URL

Create a Thomson Reuters Datastream connection. Then, retrieve the closing prices for a security. The data you see when completing this example can differ from the output data shown.

Connect to Thomson Reuters Datastream using a user name, password, data source, and URL. `c` is the Thomson Reuters Datastream connection object.

```
username = 'DS:USER1';
password = 'XXXXXXX';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';
c = datastream(username,password,source,url)
```

```
c =

  datastream with properties:

         user: 'DS:USER1''
     password: 'XXXXXXX'
   datasource: 'Datastream'
     endpoint: 'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx'
         wsdl: 'http://dataworks.thomson.com/dataworks/enterprise/1.0/webserviceclient.asmx?WSDL
```

Retrieve the closing prices for the IBM security. The structure `d` contains each price value in the field `P` as a cell array.

```
sec = 'IBM';

d = fetch(c,sec)
```

```
d =

  struct with fields:

           Source: 'Datastream'
       Instrument: 'IBM'
       StatusType: 'Connected'
```

```
       StatusCode: '0'
    StatusMessage: ''
              CCY: 'U$'
             DATE: {262×1 cell}
         DISPNAME: 'INTL.BUS.MCHS. (IRS)'
        FREQUENCY: 'D'
                P: {262×1 cell}
           SYMBOL: 'IBM'
```

Display the first three prices in the cell array.

```
d.P{1:3}
```

```
ans =

    '157.9'


ans =

    '160.28'


ans =

    '158.99'
```

Close the Thomson Reuters Datastream connection.

```
close(c)
```

## Tips

If the `datastream` function returns a connection error, verify that your proxy settings are correct in MATLAB. On the **Home** tab, in the **Environment** section, click **Preferences** and then click **Web**.

## Compatibility Considerations

### `datastream` object will be removed

*Warns starting in R2020a*

The `datastream` object will be removed in a future release. Use the `datastreamws` object instead. Some differences between the workflows require updates to your code.

#### Update Code

Use the `datastreamws` function to create a Datastream Web Services connection.

In prior releases, you created a `datastream` object by writing code similar to the following:

```
username = 'DS:username';
password = 'password';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
```

Now specify only the user name and password with the `datastreamws` function.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

## See Also

**Introduced in R2006a**

# close

(To be removed) Close connections to Thomson Reuters Datastream data servers

**Note** The `close` function will be removed in a future release without replacement. For details, see "Compatibility Considerations".

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Thomson Reuters Datastream connection object created with the `datastream` function. |

## Description

`close(Connect)` closes a connection to a Thomson Reuters Datastream data server.

## Compatibility Considerations

### `close` function will be removed

*Warns starting in R2020a*

The `close` function will be removed in a future release without replacement. Some differences between the workflows require updates to your code.

#### Update Code

Use the `datastreamws` function to create a Datastream Web Services connection.

In prior releases, you created a `datastream` object and closed the Thomson Reuters Datastream connection by writing code similar to the following:

```
username = 'DS:username';
password = 'password';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
close(c)
```

Now specify only the user name and password with the `datastreamws` function.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

There is no replacement functionality for the `close` function.

## See Also

`datastream`

**Introduced in R2006a**

# fetch

(To be removed) Request data from Thomson Reuters Datastream data servers

**Note** The `fetch` function will be removed in a future release. Use the `history` function instead. For details, see “Compatibility Considerations”.

## Syntax

```
data = fetch(Connect, 'Security')
data = fetch(Connect, 'Security', 'Fields')
data = fetch(Connect, 'Security', 'Fields', 'Date')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
'ToDate')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
'ToDate', 'Period')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
'ToDate', 'Period', 'Currency')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
'ToDate', 'Period', 'Currency', 'ReqFlag')
```

## Arguments

| | |
|---|---|
| `Connect` | Thomson Reuters Datastream connection object created with the `datastream` function. |
| `'Security'` | MATLAB character vector or string containing the name of a security, or cell array of character vectors or string array containing names of multiple securities. This data is in a format recognizable by the Thomson Reuters Datastream data server. |
| `'Fields'` | (Optional) MATLAB character vector, string, cell array of character vectors, or string array indicating the data fields for which to retrieve data. |
| `'Date'` | (Optional) MATLAB character vector or string indicating a specific calendar date for which you request data. |
| `'FromDate'` | (Optional) Start date for historical data. |

| | |
|---|---|
| `'ToDate'` | (Optional) End date for historical data. If you specify a value for `'ToDate'`, `'FromDate'` cannot be an empty value.<br><br>**Note** You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day. |
| `'Period'` | (Optional) Period within a date range. `Period` values are:<br>• `'d'`: daily values<br>• `'w'`: weekly values<br>• `'m'`: monthly values |
| `'Currency'` | (Optional) Currency in which `fetch` returns the data. |
| `'ReqFlag'` | (Optional) Specifies how the fetch request is processed by Datastream. The default value is `0`. |

**Note** You can enter the optional arguments `'Fields'`, `'FromDate'`, `'ToDate'`, `'Period'`, and `'Currency'` as MATLAB character vectors, strings, or empty arrays (`[ ]`).

## Description

`data = fetch(Connect, 'Security')` returns the default time series for the indicated security.

`data = fetch(Connect, 'Security', 'Fields')` returns data for the specified security and fields.

`data = fetch(Connect, 'Security', 'Fields', 'Date')` returns data for the specified security and fields on a particular date.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')` returns data for the specified security and fields for the indicated date range.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period')` returns instrument data for the given range with the indicated period.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency')` also specifies the currency in which to report the data.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency', 'ReqFlag')` also specifies a `ReqFlag` that determines how the request is processed by Datastream.

**Note** The Thomson Reuters Datastream interface returns all data as character vectors. For example, it returns `Price` data to the MATLAB workspace as a cell array of character vectors within the structure. There is no way to determine the data type from the Datastream interface. For details about the Thomson Reuters Datastream returned data, see *Reuters Data Support*.

## Examples

### Retrieving Time-Series Data

Return the trailing one-year price time series for the instrument `ICI`, with the default value `P` for the `'Fields'` argument using the command:

```
data = fetch(Connect, 'ICI')
```

Or the command:

```
data = fetch(Connect, 'ICI', 'P')
```

### Retrieving Opening and Closing Prices

Return the closing and opening prices for the instruments `ICI` on the date September 1, 2007.

```
data = fetch(Connect, 'ICI', {'P', 'PO'}, '09/01/2007')
```

### Retrieving Monthly Opening and Closing Prices for a Specified Date Range

Return the monthly closing and opening prices for the securities `ICI` and `IBM` from 09/01/2005 to 09/01/2007:

```
data = fetch(Connect, {'ICI', 'IBM'}, {'P', 'PO'}, ...
'09/01/2005', '09/01/2007', 'M')
```

### Retrieving Static Data

Return the static fields `NAME` and `ISIN`:

```
data = fetch(Connect,{'IBM~REP'}, {'NAME','ISIN'});
```

You can also return `SECD` in this way.

### Retrieving Russell 1000 Constituent List

Return the Russell 1000 Constituent List:

```
russell = fetch(Connect, {'LFRUSS1L~LIST~#UserName'});
```

where `UserName` is the user name for the Thomson Reuters Datastream connection.

## Compatibility Considerations

### `fetch` function will be removed

*Warns starting in R2020a*

The `fetch` function will be removed in a future release. Use the `history` function instead. Some differences between the workflows require updates to your code.

#### Update Code

Use the `history` function to retrieve Datastream Web Services historical data.

In prior releases, you created a `datastream` object and retrieved data by writing code similar to the following:

```
username = 'DS:username';
password = 'password';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
data = fetch(Connect, 'ICI', {'P', 'PO'}, '09/01/2007');
```

Now use the `datastreamws` and `history` functions instead.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);

sec = 'VOD';
d = history(c,sec);
```

## See Also

`close` | `datastream` | `get` | `isconnection`

**Introduced in R2006a**

# get

(To be removed) Retrieve properties of Thomson Reuters Datastream connection objects

**Note** The `get` function will be removed in a future release without replacement. For details, see "Compatibility Considerations".

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Thomson Reuters Datastream connection object created with the `datastream` function. |
| `PropertyName` | (Optional) A MATLAB character vector, string, cell array of character vectors, or string array containing property names. Valid property names include:<br>• `user`<br>• `datasource`<br>• `endpoint`<br>• `wsdl`<br>• `sources`<br>• `systeminfo`<br>• `version` |

## Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the Thomson Reuters Datastream connection object.

`value = get(Connect)` returns a MATLAB structure where each field name is the name of a property of `Connect`. Each field contains the value of the property.

## Compatibility Considerations

**get function will be removed**
*Warns starting in R2020a*

The `get` function will be removed in a future release without replacement. Some differences between the workflows require updates to your code.

#### Update Code

Use the `datastreamws` function to create a Datastream Web Services connection.

In prior releases, you created a `datastream` object and retrieved properties by writing code similar to the following:

```
username = 'DS:username';
password = 'password';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
value = get(c);
```

Now specify only the user name and password with the `datastreamws` function.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

There is no replacement functionality for the `get` function. To access the properties of the `datastreamws` object, use dot notation.

## See Also

`close` | `datastream` | `fetch` | `isconnection`

#### Introduced in R2006a

# isconnection

(To be removed) Determine if connections to Thomson Reuters Datastream data servers are valid

**Note** The `isconnection` function will be removed in a future release without replacement. For details, see "Compatibility Considerations".

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Thomson Reuters Datastream connection object created with the `datastream` function. |

## Description

`x = isconnection(Connect)` returns `x = 1` if the connection is a valid Thomson Reuters Datastream connection, and `x = 0` otherwise.

## Examples

Establish a connection to the Thomson Reuters Datastream API:

```
c = datastream
```

Verify that `c` is a valid connection:

```
x = isconnection(c)
x = 1
```

## Compatibility Considerations

### `isconnection` function will be removed

*Warns starting in R2020a*

The `isconnection` function will be removed in a future release without replacement. Some differences between the workflows require updates to your code.

#### Update Code

Use the `datastreamws` function to create a Datastream Web Services connection.

In prior releases, you created a `datastream` object and verified the connection by writing code similar to the following:

```
username = 'DS:username';
password = 'password';
```

```
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
x = isconnection(c);
```

Now specify only the user name and password with the `datastreamws` function.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

There is no replacement functionality for the `isconnection` function.

## See Also

`close` | `datastream` | `fetch` | `get`

**Introduced in R2006a**

# ihsmarkitrs

IHS Markit connection

# Description

The `ihsmarkitrs` function creates an `ihsmarkitrs` object, which represents an IHS Markit connection. First, you must obtain credentials from IHS Markit. For credentials, see the IHS Markit website. After you create an `ihsmarkitrs` object, you can use the object functions to retrieve factor, security, signal, and universe information.

# Creation

## Syntax

```
c = ihsmarkitrs(username,password)
```

### Description

`c = ihsmarkitrs(username,password)` creates an IHS Markit connection using a password and sets the Username property.

### Input Arguments

**`password` — Password**
character vector | string scalar

Password, specified as a character vector or string scalar. For credentials, see the IHS Markit website.

Data Types: `char` | `string`

## Properties

**`Username` — User name**
character vector | string scalar

User name, specified as a character vector or string scalar. For credentials, see the IHS Markit website.

Data Types: `char` | `string`

**`TimeOut` — Timeout**
`100` (default) | numeric scalar

Timeout, specified as a numeric scalar that indicates the number of seconds to wait for data to return before canceling the request.

Example: `10`

Data Types: `double`

## Object Functions

| | |
|---|---|
| factorgroups | Retrieve factor group information |
| factors | Retrieve factor information |
| security | Retrieve security information |
| signals | Retrieve signal information |
| universes | Retrieve universe information |

## Examples

### Retrieve Factor Groups

Using an IHS Markit connection, retrieve factor groups.

Create an IHS Markit connection using your user name and password.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password)
```

```
c =

  ihsmarkitrs with properties:

    Username: 'ABCDEF'
     TimeOut: 100
```

`c` is an `ihsmarkitrs` object with the `Username` and `TimeOut` properties. The appearance of the `ihsmarkitrs` object indicates a successful connection. The `Username` property contains your IHS Markit user name. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return factor group data before canceling the request.

Retrieve factor group data using the IHS Markit connection. `d` is a table that contains factor group information.

```
d = factorgroups(c);
```

Display the first few rows of factor group data.

```
head(d)
```

```
ans =

  8×5 table

    id              name              hasCompositeModels    hasSubCompositeModels    hasBasicFac
    __    ____________________________    __________________    _____________________    ___________

     2    'Deep Value'                          true                  true                  true
     3    'Earnings Momentum'                   true                  true                  true
     4    'Earnings Quality'                    true                  true                  true
     5    'Historical Growth'                   true                  true                  true
     7    'Liquidity, Risk & Size'              true                  true                  true
     8    'Management Quality'                  true                  true                  true
     9    'Price Momentum'                      true                  true                  true
    10    'Relative Value'                      true                  true                  true
```

d contains these variables:

- Identification number of the factor group
- Name of the factor group
- Whether the factor group contains composite factors
- Whether the factor group contains subcomposite factors
- Whether the factor group contains basic factors

## See Also

**Topics**
“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

**External Websites**
IHS Markit
IHS Markit Research Signals REST Documentation

**Introduced in R2018b**

# factorgroups

Retrieve factor group information

## Syntax

```
d = factorgroups(c)
```

## Description

`d = factorgroups(c)` returns factor group information using the IHS Markit connection.

## Examples

### Retrieve Factor Groups

Using an IHS Markit connection, retrieve factor groups.

Create an IHS Markit connection using your user name and password.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password)
```

```
c =

  ihsmarkitrs with properties:

    Username: 'ABCDEF'
     TimeOut: 100
```

`c` is an `ihsmarkitrs` object with the `Username` and `TimeOut` properties. The appearance of the `ihsmarkitrs` object indicates a successful connection. The `Username` property contains your IHS Markit user name. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return factor group data before canceling the request.

Retrieve factor group data using the IHS Markit connection. `d` is a table that contains factor group information.

```
d = factorgroups(c);
```

Display the first few rows of factor group data.

```
head(d)
```

```
ans =

  8×5 table

    id              name              hasCompositeModels    hasSubCompositeModels    hasBasicFac
    __    ____________________________    __________________    _____________________    ___________
```

```
     2    'Deep Value'                    true                    true                    true
     3    'Earnings Momentum'             true                    true                    true
     4    'Earnings Quality'              true                    true                    true
     5    'Historical Growth'             true                    true                    true
     7    'Liquidity, Risk & Size'        true                    true                    true
     8    'Management Quality'            true                    true                    true
     9    'Price Momentum'                true                    true                    true
    10    'Relative Value'                true                    true                    true
```

d contains these variables:

- Identification number of the factor group
- Name of the factor group
- Whether the factor group contains composite factors
- Whether the factor group contains subcomposite factors
- Whether the factor group contains basic factors

## Input Arguments

### c — IHS Markit connection

`ihsmarkitrs` object

IHS Markit connection, specified as an `ihsmarkitrs` object.

## Output Arguments

### d — Factor group information

table

Factor group information, specified as a table. The table contains information about available factor groups with these variables.

| Variable | Description | Data Type |
|---|---|---|
| `id` | Identification number of the factor group | `double` |
| `name` | Name of the factor group | cell array of character vectors |
| `hasCompositeModels` | Whether the factor group contains composite factors | `logical` |
| `hasSubCompositeModels` | Whether the factor group contains subcomposite factors | `logical` |
| `hasBasicFactors` | Whether the factor group contains basic factors | `logical` |

## See Also

`factors` | `ihsmarkitrs` | `security` | `signals` | `universes`

**Topics**

“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

**External Websites**
IHS Markit
IHS Markit Research Signals REST Documentation

**Introduced in R2018b**

# factors

Retrieve factor information

## Syntax

```
d = factors(c)
d = factors(c,code)
d = factors(c,code,requesttype)
d = factors(c,code,'HistoryDetail',universeid)
```

## Description

`d = factors(c)` returns a list of factors using an IHS Markit connection.

`d = factors(c,code)` returns factor information for the specified factor code or group name.

`d = factors(c,code,requesttype)` returns factor information based on the type of request.

`d = factors(c,code,'HistoryDetail',universeid)` returns the historical data for the specified factor within the specified universe using the `HistoryDetail` request.

## Examples

### Retrieve List of Factors

Using an IHS Markit connection, retrieve a list of factors.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve a list of factors using the IHS Markit connection. `d` is a table that contains the list of factors.

```
d = factors(c);
```

Display the first three variables to view the information for the first three factors.

```
d(1:3,1:3)
```

```
ans =

  3×3 table

    id        code                          name
   ____    __________    ____________________________________________

   7403    '3MChgGPA'    'Quarterly Change in Gross Profit to Assets'
   7404    '3MChgGPM'    'Quarterly Change in Gross Profit Margin'
      1    'ABR'         'Abnormal Return around QTR Earnings Release'
```

The variables are:

- `id` — Identification number of the factor
- `code` — Factor code
- `name` — Factor name

### Retrieve Factor Information for Specific Factor Code

Using an IHS Markit connection, retrieve information for a specific factor by using the factor code.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve information for the factor with the code `ABR` using the IHS Markit connection. The `factors` function returns `d` as a table that contains the information for the specified factor.

```
code = 'ABR';
d = factors(c,code);
```

Display the first three variables of the table.

```
d(1,1:3)
```

```
ans =

  1×3 table

    id    code                          name
    __    _____    ____________________________________________

    1     'ABR'    'Abnormal Return around QTR Earnings Release'
```

The variables are:

- `id` — Identification number of the factor
- `code` — Factor code
- `name` — Factor name

### Retrieve Factor Information for `Mapping` Request

Using an IHS Markit connection, retrieve information for a specific factor by using the factor code. Also, specify the `Mapping` request.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve information for the factor with the code `ABR` using the IHS Markit connection. Specify the `Mapping` type for the request. `d` is a table that has one variable, which lists the names of the universes that contain the specified factor.

```
code = 'ABR';
requesttype = 'Mapping';
d = factors(c,code,requesttype)
```

```
d =

  5×1 table

              universe
    ________________________________

    'QSG Bank Universe'
    'Markit US Large Cap'
    'Markit US Small Cap'
    'Markit US Total Cap'
    'US Total Cap Highly Shorted'
```

### Retrieve Factor Information for `HistoryDetail` Request

Using an IHS Markit connection, retrieve information for a specific factor by using the factor code. Also, specify the `HistoryDetail` request.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve information for the factor with the code `ACI` using the IHS Markit connection. Specify the `HistoryDetail` type for the request and the `QSG World` universe. `d` is a table that contains the historical information for the specified factor.

```
code = 'ACI';
universeid = 'QSG World';
d = factors(c,code,'HistoryDetail',universeid)
```

```
d =

  4×8 table

    code     factorId    universeId      dataType       universe      freqType      startDate
    _____    ________    __________    _____________    __________    _________    ___________

    'ACI'       2           133        'Percentile'    'QSG World'    'Daily'      '10/01/2009'
    'ACI'       2           133        'Percentile'    'QSG World'    'Monthly'    '12/30/1988'
    'ACI'       2           133        'Rawratio'      'QSG World'    'Daily'      '10/01/2009'
    'ACI'       2           133        'Rawratio'      'QSG World'    'Monthly'    '12/30/1988'
```

`d` is a table with these variables:

- `code` — Factor code

- `factorId` — Identification number for the factor code
- `universeId` — Identification number for the universe
- `dataType` — Data reporting format
- `universe` — Universe name
- `freqType` — Frequency (periodicity)
- `startDate` — Start date of the factor in the universe
- `endDate` — End date of the factor in the universe

## Input Arguments

### c — IHS Markit connection

`ihsmarkitrs` object

IHS Markit connection, specified as an `ihsmarkitrs` object.

### code — Factor code

character vector | string scalar

Factor code or group name, specified as a character vector or string scalar.

Example: `"ABR"`

Data Types: `char` | `string`

### requesttype — Request type

character vector | string scalar

Request type, specified as the value `'ModelStructure'` or `'Mapping'`. Use the `'ModelStructure'` value to return a list of the composite factors that constitute the factor specified by the `code` input argument. Use the `'Mapping'` value to return a list of the names of universes that contain the specified factor.

You can specify each value as a character vector or string scalar.

### universeid — Universe name

character vector | string scalar

Universe name, specified as a character vector or string scalar. Use `universeid` only with the `'HistoryDetail'` syntax.

Example: `'QSG World'`

Data Types: `char` | `string`

## Output Arguments

### d — Factor information

table

Factor information, returned as a table. The following table describes the variables in the returned data. Depending on the request type specified in the `requesttype` input argument or the `'HistoryDetail'` syntax, the returned table contains a subset of these variables.

| Variable Name | Description | Data Type |
|---|---|---|
| `id` | Identification number of the factor | `double` |
| `code` | Factor code | cell array of character vectors |
| `name` | Factor name | cell array of character vectors |
| `description` | Factor description | cell array of character vectors |
| `dIntl` | Localized factor description | cell array of character vectors |
| `type` | Factor type | cell array of character vectors |
| `parentCode` | Factor parent code | cell array of character vectors |
| `rankingOrder` | Rank order (descending or ascending) | cell array of character vectors |
| `isRankAvailable` | Whether rank data is available for the factor | `logical` |
| `isZscoreAvailable` | Whether z-score data is available for the factor | `logical` |
| `isRawRatioAvailable` | Whether raw ratio data is available for the factor | `logical` |
| `modelType` | Model type of the factor | cell array of character vectors |
| `groupId` | Group identifier of the factor | `double` |
| `groupName` | Name of the factor group | cell array of character vectors |
| `factorId` | Factor identifier | `double` |
| `universe` | Universe name | cell array of character vectors |
| `universeId` | Identification number of the universe | `double` |
| `dataType` | Reporting format of the data | cell array of character vectors |
| `freqType` | Frequency (or periodicity) of the data | cell array of character vectors |
| `startDate` | Start date of the factor in the universe | cell array of character vectors |
| `endDate` | End date of the factor in the universe | cell array of character vectors |
| `childId` | Code of the factor within the composite factor | `double` |
| `childCode` | Name of the factor within the composite factor | cell array of character vectors |
| `weight` | Weight of the factor within the composite factor | `double` |
| `data` | Country code | cell array of character vectors |

## See Also

`factorgroups` | `ihsmarkitrs` | `security` | `signals` | `universes`

**Topics**
“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

**External Websites**
IHS Markit
IHS Markit Research Signals REST Documentation

**Introduced in R2018b**

# security

Retrieve security information

## Syntax

```
d = security(c,universeid,startdate,enddate)
d = security(c,universeid,startdate,enddate,identifier)
```

## Description

`d = security(c,universeid,startdate,enddate)` returns security information using an IHS Markit connection, universe name, and date range.

`d = security(c,universeid,startdate,enddate,identifier)` specifies the type of security to retrieve.

## Examples

### Retrieve Security Information

Using an IHS Markit connection, retrieve security information using a date range within a specified universe.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve security information for the `US Total Cap` universe from January 1, 2017 through December 31, 2017 using the IHS Markit connection. `d` is a table that contains the security information.

```
universeid = "US Total Cap";
startdate = "2017-01-01";
enddate = "2017-12-31";
d = security(c,universeid,startdate,enddate);
```

Display the first few rows of security information.

```
head(d)
```

```
ans =

  8×7 table

      mid         startDate        endDate         cusip        sedol      ticker    quotCountry
    _________    ____________    ____________    _________    ________    ______    ___________

    1.3183e+05    '05/10/2013'    '01/01/2050'    '03265410'    '203206'    'ADI'         ''
    1.3262e+05    '05/10/2013'    '01/01/2050'    '00790310'    '200784'    'AMD'         ''
```

```
    1.3492e+05    '05/10/2013'    '01/01/2050'    '09676110'    '210775'    'BOBE'        ''
    1.4093e+05    '05/10/2013'    '01/01/2050'    '12550910'    '219647'    'CI'          ''
    1.4205e+05    '05/10/2013'    '01/01/2050'    '14428510'    '217750'    'CRS'         ''
    1.4224e+05    '05/10/2013'    '01/01/2050'    '12640810'    '216075'    'CSX'         ''
    1.4226e+05    '05/10/2013'    '01/01/2050'    '21683110'    '222260'    'CTB'         ''
    1.4344e+05    '05/10/2013'    '01/01/2050'    '24801910'    '226036'    'DLX'         ''
```

The variables are:

- `mid` — IHS Markit identification code
- `startDate` — Start date of the factor in the universe
- `endDate` — End date of the factor in the universe
- `cusip` — CUSIP security identifier
- `sedol` — SEDOL security identifier
- `ticker` — Ticker security identifier
- `quotCountry` — Market country of the security

### Retrieve Security Information for Specific Security Type

Using an IHS Markit connection, retrieve security information using a date range within a specified universe. Specify the type of security to retrieve.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve security information for the `US Total Cap` universe from January 1, 2017 through December 31, 2017 using the IHS Markit connection. Specify retrieving only SEDOL security identifiers. `d` is a table that contains the security information.

```
universeid = "US Total Cap";
startdate = "2017-01-01";
enddate = "2017-12-31";
identifier = "sedol";
d = security(c,universeid,startdate,enddate,identifier);
```

Display the first few rows of security information.

```
head(d)
```

```
ans =

  8×5 table

      mid          startDate        endDate         sedol      quotCountry
    ___________    ____________    ____________    ________    ___________

    1.3233e+05    '05/10/2013'    '01/01/2050'    '200111'        ''
      1.33e+05    '05/10/2013'    '01/01/2050'    '204617'        ''
    1.3353e+05    '05/10/2013'    '01/01/2050'    '206051'        ''
    1.3376e+05    '05/10/2013'    '01/01/2050'    '206650'        ''
```

```
    1.4304e+05    '05/10/2013'    '01/01/2050'    '227646'        ''
    1.4424e+05    '05/10/2013'    '01/01/2050'    '231380'        ''
    1.4498e+05    '05/10/2013'    '01/01/2050'    '232204'        ''
      1.46e+05    '05/10/2013'    '01/01/2050'    '234292'        ''
```

The variables are:

- `mid` — IHS Markit identification code
- `startDate` — Start date of the factor in the universe
- `endDate` — End date of the factor in the universe
- `sedol` — SEDOL security identifier
- `quotCountry` — Market country of the security

## Input Arguments

### `c` — IHS Markit connection

`ihsmarkitrs` object

IHS Markit connection, specified as an `ihsmarkitrs` object.

### `universeid` — Universe name

character vector | string scalar

Universe name, specified as a character vector or string scalar.

Example: `'US Total Cap'`

Data Types: `char` | `string`

### `startdate` — Start date

`datetime` array | numeric scalar | character vector | string scalar

Start date for a data request, specified as a `datetime` array, numeric scalar, character vector, or string scalar.

Example: `"2017-01-01"`

Data Types: `double` | `char` | `string` | `datetime`

### `enddate` — End date

`datetime` array | numeric scalar | character vector | string scalar

End date for a data request, specified as a `datetime` array, numeric scalar, character vector, or string scalar.

Example: `"2017-12-31"`

Data Types: `double` | `char` | `string` | `datetime`

### `identifier` — Security type

character vector | string scalar | cell array of character vectors | string array

Security type to retrieve, specified as one or more of these values: `'ticker'`, `'cusip'`, or `'sedol'`. You can specify these values as a character vector, string scalar, cell array of character vectors, or string array.

## Output Arguments

### d — Security information

table

Security information, returned as a table. The following table describes the variables in the returned data. (The variables for security type vary depending on the type that you specify in the `identifier` input argument.)

| Variable Name | Description | Data Type |
|---|---|---|
| `mid` | IHS Markit identification code | `double` |
| `startDate` | Start date of the factor in the universe | cell array of character vectors |
| `endDate` | End date of the factor in the universe | cell array of character vectors |
| `cusip` | CUSIP security identifier | cell array of character vectors |
| `ticker` | Ticker security identifier | cell array of character vectors |
| `sedol` | SEDOL security identifier | cell array of character vectors |
| `quotCountry` | Market country of the security | cell array of character vectors |

## See Also

`factorgroups` | `factors` | `ihsmarkitrs` | `signals` | `universes`

### Topics

“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

### External Websites

IHS Markit
IHS Markit Research Signals REST Documentation

### Introduced in R2018b

# signals

Retrieve signal information

## Syntax

```
d = signals(c,code,universeid,startdate,enddate)
d = signals(c,code,universeid,startdate,enddate,identifier)
d = signals(c,code,universeid,startdate,enddate,identifier,datatype)
d = signals(c,code,universeid,startdate,enddate,identifier,datatype,
monthlydata)
```

## Description

`d = signals(c,code,universeid,startdate,enddate)` returns signal information using an IHS Markit connection, factor code, universe name, and date range.

`d = signals(c,code,universeid,startdate,enddate,identifier)` specifies the type of security to retrieve.

`d = signals(c,code,universeid,startdate,enddate,identifier,datatype)` specifies the data format for the returned signal information.

`d = signals(c,code,universeid,startdate,enddate,identifier,datatype, monthlydata)` specifies retrieval of monthly data.

## Examples

### Retrieve Signal Information

Using an IHS Markit connection, retrieve signal information using a factor and date range within a specified universe.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve signal information for the last 10 days using the IHS Markit connection. Specify the `ABR` factor code and `QSG World` universe. `ABR` is a sample factor code and `QSG World` is a sample universe. To retrieve signal information for your code and universe combination, substitute the factor code in `code` and universe in `universeid`. The `d` workspace variable is a table that contains signal information and the `date` and `data` variables.

```
code = 'ABR';
universeid = 'QSG World';
startdate = datetime('today')-10;
enddate = datetime('today');
d = signals(c,code,universeid,startdate,enddate);
```

Access the first few rows of signal information for the first day in the date range by using the `data` variable.

```
data = d.data{1};
head(data)
```

```
ans =

  8×2 table

    ticker    value
    ______    _____

    'VIRT'      1
    'SEDG'      1
    'CRTO'      1
    'BZUN'      1
    'FNGN'      1
    'CMG'       1
    'INGN'      1
    'ADAP'      1
```

The variables of the resulting table are `ticker` and `value`. The `ticker` variable contains the ticker security identifiers. The `value` variable contains the signal information for the corresponding security.

### Retrieve Signal Information for SEDOL Security Type

Using an IHS Markit connection, retrieve signal information using a factor and date range within a specified universe. Specify the SEDOL security type.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve signal information for the last 10 days using the IHS Markit connection. Specify the `ABR` factor code and `QSG World` universe. `ABR` is a sample factor code and `QSG World` is a sample universe. To retrieve signal information for your code and universe combination, substitute the factor code in `code` and universe in `universeid`. Also, specify the SEDOL security type. `d` is a table that contains signal information and the `date` and `data` variables.

```
code = 'ABR';
universeid = 'QSG World';
startdate = datetime('today')-10;
enddate = datetime('today');
identifier = 'sedol';
d = signals(c,code,universeid,startdate,enddate,identifier);
```

Access the first few rows of signal information for the first day in the date range by using the `data` variable.

```
data = d.data{1};
head(data)
```

```
ans =

  8×2 table

     sedol      value
    ________    _____

    'BWTVWD'      1
    'BWC52Q'      1
    'BFPMB2'      1
    'BY2ZJ6'      1
    'B65V2X'      1
    'B0X7DZ'      1
    'BJSVLL'      1
    'BWY4XV'      1
```

The variables of the resulting table are `sedol` and `value`. The `sedol` variable contains the SEDOL security identifiers. The `value` variable contains the signal information for the corresponding security.

### Retrieve Signal Information for Z-Score Data Format

Using an IHS Markit connection, retrieve signal information using a factor and date range within a specified universe. Specify the SEDOL security type and z-score data format.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve signal information for the last 10 days using the IHS Markit connection. Specify the `ABR` factor code and `QSG World` universe. `ABR` is a sample factor code and `QSG World` is a sample universe. To retrieve signal information for your code and universe combination, substitute the factor code in `code` and universe in `universeid`. Also, specify the SEDOL security type and z-score data format. `d` is a table that contains signal information and the `date` and `data` variables.

```
code = 'ABR';
universeid = 'QSG World';
startdate = datetime('today')-10;
enddate = datetime('today');
identifier = 'sedol';
datatype = 'zscore';
d = signals(c,code,universeid,startdate,enddate,identifier,datatype);
```

Access the first few rows of signal information for the first day in the date range by using the `data` variable.

```
data = d.data{1};
head(data)
```

```
ans =

  8×2 table

     sedol       value
```

```
    ________    ________

    'B44WZD'     0.63461
    'B4MG4Z'     0.43807
    '281355'     -3.3183
    'BF4VWH'     0.94079
    'B92SR7'     0.80995
    'BWY4XV'      3.1591
    'B1VZ43'    -0.25296
    '236542'    -0.77368
```

The variables of the resulting table are `sedol` and `value`. The `sedol` variable contains the SEDOL security identifiers. The `value` variable contains the signal information for the corresponding security as a z-score.

### Retrieve Monthly Signal Information for Specific Data Format

Using an IHS Markit connection, retrieve monthly signal information using a factor and date range within a specified universe. Specify the SEDOL security type and z-score data format.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve signal information for the last 3 months using the IHS Markit connection. Specify the `ABR` factor code and `QSG World` universe. `ABR` is a sample factor code and `QSG World` is a sample universe. To retrieve signal information for your code and universe combination, substitute the factor code in `code` and universe in `universeid`. Also, specify the SEDOL security type and z-score data format. `d` is a table that contains signal information and the `date` and `data` variables.

```
code = 'ABR';
universeid = 'QSG World';
startdate = datetime('today')-90;
enddate = datetime('today');
identifier = 'sedol';
datatype = 'zscore';
monthlydata = 'true';
d = signals(c,code,universeid,startdate,enddate, ...
    identifier,datatype,monthlydata);
```

Access the first few rows of signal information for the first month in the date range by using the `data` variable.

```
data = d.data{1};
head(data)
```

```
ans =

  8×2 table

     sedol       value
    ________    ________

    'B44WZD'     0.44178
```

```
    'B4MG4Z'     -1.2075
    '281355'     0.43517
    'BF4VWH'     0.91456
    'B92SR7'       2.065
    '256652'     0.49538
    'B1VZ43'    -0.26471
    'BFRTDG'    -0.69078
```

The variables of the resulting table are `sedol` and `value`. The `sedol` variable contains the SEDOL security identifiers. The `value` variable contains the signal information for the corresponding security as a z-score.

## Input Arguments

### `c` — IHS Markit connection

`ihsmarkitrs` object

IHS Markit connection, specified as an `ihsmarkitrs` object.

### `code` — Factor code

character vector | string scalar

Factor code, specified as a character vector or string scalar.

Example: `"ABR"`

Data Types: `char` | `string`

### `universeid` — Universe name

character vector | string scalar

Universe name, specified as a character vector or string scalar.

Example: `'US Total Cap'`

Data Types: `char` | `string`

### `startdate` — Start date

`datetime` array | numeric scalar | character vector | string scalar

Start date for a data request, specified as a `datetime` array, numeric scalar, character vector, or string scalar.

Example: `"2017-01-01"`

Data Types: `double` | `char` | `string` | `datetime`

### `enddate` — End date

`datetime` array | numeric scalar | character vector | string scalar

End date for a data request, specified as a `datetime` array, numeric scalar, character vector, or string scalar.

Example: `"2017-12-31"`

Data Types: `double` | `char` | `string` | `datetime`

### `identifier` — Security type

`"ticker"` (default) | character vector | string scalar | cell array of character vectors | string array

Security type to retrieve, specified as one or more of these values: `'ticker'`, `'cusip'`, or `'sedol'`. You can specify these values as a character vector, string scalar, cell array of character vectors, or string array.

**`datatype` — Data format**
`"percentile"` (default) | character vector | string scalar

Data format, specified as one of these values.

| Data Format Value | Description | Calculation |
|---|---|---|
| `"percentile"` | Percentile rank (from `1` through `100`) of the factor | Rank the securities in the universe into percentiles, by using the factor value, in ascending or descending order based on the definition. The `signals` function ranks the securities with the most attractive value as `1` and securities with the least attractive value as `100`. |
| `"rawratio"` | Raw value of the factor | The numeric output of the factor calculation. |
| `"rawrank"` | Ordinal rank (from `1` through *n*) of the factor | Rank the securities in the universe in ordinal order, by using the factor value, in ascending or descending order based on the definition. |
| `"zscore"` | Z-score of the factor | Determine the mean and standard deviation of all factor values in the universe on the specified date. Then, subtract the mean from the factor value of the security and divide the result by the standard deviation. |

You can specify each value as a character vector or string scalar.

**`monthlydata` — Monthly indicator**
`"false"` (default) | character vector | string scalar

Monthly indicator, specified as the value `"true"` or `"false"`. When the `monthlydata` input argument is `"true"`, the `signals` function returns monthly data. Otherwise, the `signals` function returns daily data.

## Output Arguments

**`d` — Signal information**
table

Signal information, returned as a table with the `date` and `data` variables. The `date` variable contains each date in the specified date range. If you specify monthly data using the `monthlydata` input

argument, then the `date` variable contains one row for each month. The `data` variable contains a table of data for each corresponding date. To access the data for the first day in the date range, use dot notation, for example: `d.data{1}`.

## See Also

`factorgroups` | `factors` | `ihsmarkitrs` | `security` | `universes`

**Topics**

“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

**External Websites**

IHS Markit
IHS Markit Research Signals REST Documentation

**Introduced in R2018b**

# universes

Retrieve universe information

## Syntax

```
d = universes(c)
d = universes(c,universeid)
d = universes(c,universeid,requesttype)
```

## Description

`d = universes(c)` returns universe information for all universes using an IHS Markit connection.

`d = universes(c,universeid)` returns universe information for a specific universe.

`d = universes(c,universeid,requesttype)` returns universe information based on the request type.

## Examples

### Retrieve Universe Information for All Universes

Using an IHS Markit connection, retrieve universe information for all universes.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve universe information for all universes using the IHS Markit connection. `d` is a table that contains the universe information.

```
d = universes(c);
```

Display information for the first few universes.

```
head(d)
```

```
ans =

  8×6 table

    description               region                 universeType    identifier    universeId
    ___________    ____________________________    ____________    __________    __________

        ''         'Europe, Middle East & Africa'      'Global'         []            248
        ''         '11,12,13,14'                       'Global'         []            227
        ''         '11,12'                             'Global'         []            250
        ''         'Far East'                          'Global'         []            249
        ''         'Far East'                          'Global'         []            228
```

```
        ''          '11,12,13,14'                    'UDM'             []             324
        ''          '11,12,13,14'                    'Global'          []            1552
        ''          '11,12,13,14'                    'Global'          []            1293
```

The variables are:

- `description` — Universe description
- `region` — Country or region code
- `universeType` — Universe type
- `identifier` — Identification type
- `universeId` — Universe identifier
- `universe` — Universe name

### Retrieve Universe Information for Specific Universe

Using an IHS Markit connection, retrieve universe information for a specific universe.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve universe information for the `QSG World` universe using the IHS Markit connection. `d` is a table that contains the universe information.

```
universeid = "QSG World";
d = universes(c,universeid)
```

```
d =

  1×6 table

    description       region        universeType    identifier    universeId     universe
    ___________    _____________    ____________    __________    __________    ___________

        ''         '11,12,13,14'      'Global'       'Sedol'         133        'QSG World'
```

The variables are:

- `description` — Universe description
- `region` — Country or region code
- `universeType` — Universe type
- `identifier` — Identification type
- `universeId` — Universe identifier
- `universe` — Universe name

### Retrieve Universe Information for Specific Request Type

Using an IHS Markit connection, retrieve universe information for a specific universe. Specify a historical request to retrieve historical information for the universe.

Create an IHS Markit connection using your user name and password. `c` is an `ihsmarkitrs` object.

```
username = 'ABCDEF';
password = 'ABC123';
c = ihsmarkitrs(username,password);
```

Retrieve universe information for the `QSG World` universe using the IHS Markit connection. Specify retrieving historical information by using the `HistoryDetail` request type. `d` is a table that contains the historical universe information.

```
universeid = "QSG World";
requesttype = 'HistoryDetail';
d = universes(c,universeid,requesttype)
```

```
d =

  2×4 table

     universe      freqType      startDate        endDate
    ___________    _________    ____________    ____________

    'QSG World'    'Daily'      '03/22/2007'    '03/14/2018'
    'QSG World'    'Monthly'    '12/30/1988'    '05/31/2017'
```

The variables are:

- `universe` — Universe name
- `freqType` — Data frequency (or periodicity)
- `startDate` — Start date of the life of the universe
- `endDate` — End date of the life of the universe

## Input Arguments

### c — IHS Markit connection

`ihsmarkitrs` object

IHS Markit connection, specified as an `ihsmarkitrs` object.

### universeid — Universe name

character vector | string scalar

Universe name, specified as a character vector or string scalar.

Example: `'US Total Cap'`

Data Types: `char` | `string`

### requesttype — Request type

character vector | string scalar

Request type, specified as the value `'HistoryDetail'`, `'Mapping'`, or `'Country'`. Use the `'HistoryDetail'` value to return historical information from the universe that you specify using the

`universeid` input argument. Use the `'Mapping'` value to return a list of the factors in the specified universe. Use the `'Country'` value to return the country identifiers that apply to the specified universe.

You can specify each value as a character vector or string scalar.

## Output Arguments

### d — Universe information

table

Universe information, returned as a table. The following table describes the variables in the returned data. (The variables vary depending on the request type that you specify in the `requesttype` input argument.)

| Variable Name | Description | Data Type |
|---|---|---|
| `universeId` | Universe identifier | `double` |
| `description` | Universe description | cell array of character vectors |
| `region` | Country or region code | cell array of character vectors |
| `universeType` | Universe type | cell array of character vectors |
| `identifier` | Identification type | cell array of character vectors |
| `universe` | Universe name | cell array of character vectors |
| `freqType` | Data frequency (or periodicity) | cell array of character vectors |
| `startDate` | Start date of the life of the universe | cell array of character vectors |
| `endDate` | End date of the life of the universe | cell array of character vectors |
| `data` | Factors in the universe | structure |
| `country` | Countries that apply to the universe | cell array of character vectors |

## See Also

`factorgroups` | `factors` | `ihsmarkitrs` | `security` | `signals`

### Topics

“Retrieve Factor Rank Data for Portfolio Selection” on page 11-2
“IHS Markit Error Messages” on page 11-4

### External Websites

IHS Markit
IHS Markit Research Signals REST Documentation

### Introduced in R2018b

# iqf

IQFEED Desktop API connection

## Description

The `iqf` function creates an `iqf` object. The `iqf` object represents an IQFEED Desktop API connection.

After you create an `iqf` object, you can use the object functions to retrieve intraday, historical, and news data. You can also retrieve level 1 and 2 data.

## Creation

### Syntax

```
c = iqf(username,password)
c = iqf(username,password,portname)
```

#### Description

`c = iqf(username,password)` starts IQFEED or makes a connection to an existing IQFEED session, and sets the "User" on page 12-0 and "Password" on page 12-0 properties.

`c = iqf(username,password,portname)` uses a port identifier for the IQFEED connection.

**Note** Only one IQFEED connection can be open at a time.

#### Input Arguments

**`portname` — Port identifier**
`'Admin'` (default) | character vector | string scalar

Port identifier for the socket connection, specified as a character vector or string scalar.

Data Types: `char` | `string`

### Properties

**`User` — User name**
character vector | string scalar

User name, specified as a character vector or string scalar. For credentials, contact IQFEED.

Example: `'username'`

Data Types: `char` | `string`

**`Password` — Password**
character vector | string scalar

Password, specified as a character vector or string scalar. For credentials, contact IQFEED.

Example: `'pwd'`

Data Types: `char` | `string`

**`Port` — Port**
cell array

Port, specified as a cell array that contains a Microsoft .NET Framework Socket object.

Example: `{[1×1 System.Net.Sockets.Socket]}`

Data Types: `cell`

**`PortName` — Port identifier**
`{'Admin'}` (default) | cell array

Port identifier, specified as a cell array that contains a character vector or string scalar. The text specifies the port for the socket connection.

The `iqf` function sets this property using the `portname` input argument.

Data Types: `cell`

**`Protocol` — Protocol**
`[]` (default) | double

Protocol, specified as a double. The protocol specifies the IQFEED version, which determines the format for the return data.

When you create an `iqf` object, the `iqf` function leaves this property unset. Set this value manually at the command line using dot notation. For example, to set the protocol to IQFEED version 5.1, enter:

```
c.Protocol = 5.1;
```

Data Types: `double`

## Object Functions

| | |
|---|---|
| close | Close IQFEED ports |
| history | IQFEED asynchronous historical end-of-period data |
| news | IQFEED asynchronous news data |
| marketdepth | IQFEED asynchronous level 2 data |
| realtime | IQFEED asynchronous level 1 data |
| timeseries | IQFEED asynchronous intraday tick data |

## Examples

### Connect to IQFEED

Create an IQFEED connection. Then, retrieve historical daily data for a security. The historical data you see when completing this example can differ from the output data shown.

Create an IQFEED connection with the user name `username` and password `pwd`. The `iqf` object `c` appears in the MATLAB workspace.

```
username = 'username';
password = 'pwd';
c = iqf(username,password)
```

```
c =

  iqf with properties:

        User: 'username'
    Password: 'pwd'
        Port: {[1×1 System.Net.Sockets.Socket]}
    PortName: {'Admin'}
    Protocol: []
```

Retrieve the Google security data for the last five days.

```
history(c,'GOOG',5)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:08:58'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033.56'    '1277772'    '0'
```

Each row of data represents one day. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

### Connect to IQFEED Using Port Identifier

Create an IQFEED connection using a port identifier. Then, retrieve historical daily data for a security. The historical data you see when completing this example can differ from the output data shown.

Create an IQFEED connection with the user name `username`, password `pwd`, and port identifier `Admin`. The `iqf` object `c` appears in the MATLAB workspace.

```
username = 'username';
password = 'pwd';
portname = 'Admin';
c = iqf(username,password,portname)
```

```
c =

  iqf with properties:

       User: 'username'
   Password: 'pwd'
       Port: {[1×1 System.Net.Sockets.Socket]}
   PortName: {'Admin'}
   Protocol: []
```

Retrieve the Google security data for the last five days.

```
history(c,'GOOG',5)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:08:58'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033.56'    '1277772'    '0'
```

Each row of data represents one day. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

## See Also

**Topics**
“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24

**External Websites**
IQFEED Developers

**Introduced in R2012b**

# close

Close IQFEED ports

## Syntax

```
close(Q)
```

## Description

`close(Q)` closes all IQFEED ports currently open for a given IQFEED connection handle, `Q`.

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf`. |

## Examples

Close all ports for an IQFEED connection handle.

```
close(Q)
```

## See Also

`iqf`

**Topics**

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24

**Introduced in R2012b**

# history

IQFEED asynchronous historical end-of-period data

## Syntax

```
history(c,s,interval)
history(c,s,interval,period)
history(c,s,interval,period,listener,eventhandler)

history(c,s,{startdate,enddate})
history(c,s,{startdate,enddate},[],listener,eventhandler)
```

## Description

`history(c,s,interval)` returns asynchronous historical end-of-period data using the connection object `c`, a single security `s`, and a specified interval `interval`.

`history(c,s,interval,period)` returns asynchronous historical end-of-period data for a single security with a specified interval and period `period`.

`history(c,s,interval,period,listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a specified interval, period, socket listener `listener`, and event handler `eventhandler`.

`history(c,s,{startdate,enddate})` returns asynchronous historical end-of-period data for a single security with a date range.

`history(c,s,{startdate,enddate},[],listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a date range, a specified socket listener `listener`, and event handler `eventhandler`.

## Examples

### Retrieve Daily Data

Create the IQFEED connection with user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

Retrieve the Google security data for the last five days.

```
history(c,'GOOG',5)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =
```

```
    '2013-11-21 11:08:58'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033.56'    '1277772'    '0'
```

Each row of data represents one day. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

### Retrieve Weekly Data

Create the IQFEED connection with user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

Retrieve the Google security data for the last five weeks.

```
history(c,'GOOG',5,'Weekly')
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:07:02'    '1048.74'    '1020.36'    '1035.75'    '1034.07'    '4949900'     '0'
    '2013-11-15 11:07:02'    '1039.75'    '1005.00'    '1009.51'    '1033.56'    '6361983'     '0'
    '2013-11-08 11:07:02'    '1032.37'    '1007.64'    '1031.50'    '1016.03'    '6209876'     '0'
    '2013-11-01 11:07:02'    '1041.52'    '1012.98'    '1015.20'    '1027.04'    '7025769'     '0'
    '2013-10-25 11:07:02'    '1040.57'    '995.79'     '1011.46'    '1015.20'    '12636223'    '0'
```

Each row of data represents the last day of a week. The first row contains data for the last business day in the current week. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

### Retrieve Monthly Data with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

Retrieve the Google security data for the last five months. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data.

```
history(c,'GOOG',5,'Monthly',@iqhistoryfeedlistener,...
        @iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:13:07'    '1048.74'    '1005.00'    '1031.79'    '1034.07'    '18805697'    '0'
    '2013-10-31 11:13:07'    '1041.52'    '842.98'     '880.25'     '1030.58'    '55288774'    '0'
    '2013-09-30 11:13:07'    '905.99'     '853.95'     '854.36'     '875.91'     '33147210'    '0'
    '2013-08-30 11:13:07'    '909.71'     '845.56'     '895.00'     '846.90'     '33509358'    '0'
    '2013-07-31 11:13:07'    '928.00'     '875.61'     '886.45'     '887.75'     '51277966'    '0'
```

Each row of data represents the last day of a month. The first row contains data for the last business day in the current month. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

### Retrieve Data for a Date Range

Create the IQFEED connection with user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

Retrieve IBM security data for the last five days.

```
history(c,'IBM',{floor(now-5),floor(now)})
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 10:59:51'    '185.7500'    '183.4110'    '185.5400'    '184.1300'    '4459451'    '0'
    '2013-11-20 10:59:51'    '186.2400'    '184.6450'    '185.2200'    '185.1900'    '3646117'    '0'
    '2013-11-19 10:59:51'    '186.2000'    '184.1500'    '184.6300'    '185.2500'    '4577037'    '0'
    '2013-11-18 10:59:51'    '184.9900'    '183.2700'    '183.5200'    '184.4700'    '5344864'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

### Retrieve Data for a Date Range with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

```
c = iqf('username','pwd');
```

Retrieve the Google security data for the last five days. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data. The period `[]` specifies the default period for daily data.

```
history(c,'GOOG',{floor(now-5),floor(now)},[],...
        @iqhistoryfeedlistener,@iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:12:15'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:12:15'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:12:15'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:12:15'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

- Date and time
- High price
- Low price
- Open price
- Closing price
- Volume
- Open interest

Close the IQFEED connection.

```
close(c)
```

## Input Arguments

### `c` — IQFEED connection

connection object

IQFEED connection, specified as a connection object created using `iqf`.

### `s` — Security

character vector | string scalar

Security, specified as a character vector or string scalar for a single security.

Example: `'IBM'`

Data Types: `char` | `string`

### `interval` — Time interval

numeric scalar

Time interval, specified as a numeric scalar to denote the number of days of data to return.

Data Types: `double`

### `period` — Period

`'Daily'` (default) | `'Weekly'` | `'Monthly'`

Period, specified as one of the preceding values to denote daily, weekly, or monthly return data. When this argument is specified along with `interval`, `history` returns the number of daily, weekly, or monthly data where the number of output rows corresponds to the `interval`. When this argument is omitted by specifying `[]`, `history` returns daily data.

### `listener` — Listener event handler

function

Listener event handler, specified as a function to listen for the IQFEED data. You can modify the existing listener function or define your own. You can find the code for the existing listener function in the `history.m` file.

Data Types: `function_handle`

### `eventhandler` — Event handler

function

Event handler, specified as a function to process the IQFEED data. The existing event handler displays the IQFEED data in the Command Window. You can modify the existing event handler function or write your own. You can find the code for the existing event handler function in the `history.m` file.

**Note** The `history` function uses the same port as the `timeseries` function. These functions return data in either the `IQFeedTimeseriesData` or `IQFeedHistoryData` MATLAB workspace variable created by the first executed function. For different behavior, write an event handler function to process the returned data. For details, see "Writing and Running Custom Event Handler Functions" on page 1-26.

Data Types: `function_handle`

**`startdate` — Start date**
numeric scalar | character vector | string scalar

Start date, specified as a numeric scalar, character vector, or string scalar to denote the start date of the date range for the returned data.

Example: `floor(now-1)`

Data Types: `double` | `char` | `string`

**`enddate` — End date**
numeric scalar | character vector | string scalar

End date, specified as a numeric scalar, character vector, or string scalar to denote the end date of the date range for the returned data.

Example: `floor(now)`

Data Types: `double` | `char` | `string`

## Tips

- When you make multiple requests with multiple messages, this error might occur:

  Warning: Error occurred while executing delegate callback: Message: The IAsyncResult object was not returned from the corresponding asynchronous method on this class.

  To fix this, restart MATLAB.

## See Also

`close` | `iqf` | `marketdepth` | `realtime` | `timeseries`

**Topics**
"Retrieve Intraday and Historical Data Using IQFEED" on page 1-24
"Writing and Running Custom Event Handler Functions" on page 1-26

**Introduced in R2012b**

# marketdepth

IQFEED asynchronous level 2 data

## Syntax

```
marketdepth(Q,S)
marketdepth(Q,S,elistener,ecallback)
```

## Description

`marketdepth(Q,S)` returns asynchronous level 2 data using the default socket listener and event handler.

`marketdepth(Q,S,elistener,ecallback)` returns asynchronous level 2 data using an explicitly defined socket listener and event handler.

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf`. |
| `S` | `S` is specified as a character vector or string for a single security or a cell array of character vectors or string array for multiple securities. |
| `elistener` | Function handle that specifies the function used to listen for data on the level 2 port. |
| `ecallback` | Function handle that specifies the function that processes data event. |

## Examples

Return level 2 data using the default socket listener and event handler and display the results in the MATLAB workspace in the variable `IQFeedLevelTwoData`.

```
marketdepth(q,'ABC')
openvar('IQFeedLevelTwoData')
```

Initiate a watch on the security ABC for level 2 data using the function handles `iqfeedlistener` and `iqfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedLevelTwoData`.

```
marketdepth(q,'ABC',@iqfeedmarketdepthlistener,@iqfeedmarketdepteventhandler)
openvar('IQFeedLevelTwoData')
```

## See Also

`close` | `history` | `iqf` | `realtime` | `timeseries`

**Topics**

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24
“Writing and Running Custom Event Handler Functions” on page 1-26

**Introduced in R2012b**

# news

IQFEED asynchronous news data

## Syntax

```
news(Q,S)
news(Q,S,elistener,ecallback)
```

## Description

`news(Q,S)` returns asynchronous news data using the default socket listener and event handler.

`news(Q,S,elistener,ecallback)` returns asynchronous news data using an explicitly defined socket listener and event handler.

The syntax `news(Q,true)` turns on news updates for the list of currently subscribed level 1 securities and `news(Q,false)` turns off news updates for the list of currently subscribed level 1 securities.

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf`. |
| `S` | `S` is specified as a character vector or string for a single security or a cell array of character vectors or string array for multiple securities. |
| `elistener` | Function handle that specifies the function used to listen for data on the news lookup port. |
| `ecallback` | Function handle that specifies the function that processes data events. |

## Examples

Return news data using the defaults for socket listener and event handler and display the results in the MATLAB workspace in the variable `IQFeedNewsData`.

```
news(q,'ABC')
openvar('IQFeedNewsData')
```

Return news data for the security ABC using the function handles `iqfeedlistener` and `iqfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedNewsData`.

```
news(q,'ABC',@iqfeednewslistener,@iqfeednewseventhandler)
openvar('IQFeedNewsData')
```

## See Also

`close` | `history` | `iqf` | `marketdepth` | `realtime` | `timeseries`

**Topics**

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24
“Writing and Running Custom Event Handler Functions” on page 1-26

**Introduced in R2012b**

# realtime

IQFEED asynchronous level 1 data

## Syntax

```
realtime(Q,S)
realtime(Q,S,F)
realtime(Q,S,elistener,ecallback)
```

## Description

`realtime(Q,S)` returns asynchronous level 1 data using the current update field list, default socket listener, and event handler.

`realtime(Q,S,F)` returns asynchronous level 1 data for a specified field list using the default socket listener and event handler.

`realtime(Q,S,elistener,ecallback)` returns asynchronous level 1 data using an explicitly defined socket listener and event handler. For example, you can return this data for security `ABC` using the function handles `iqfeedlistener` and `iqfeedeventhandler`.

```
realtime(q,'ABC', ...
      {'Symbol','Exchange ID','Last','Change','Incremental Volume'}, ...
      @iqfeedlistener,@iqfeedeventhandler)
```

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf`. |
| `S` | `S` is specified as a character vector or string for a single security or a cell array of character vectors or string array for multiple securities. |
| `F` | `F` is the field list. If no field list is specified or it is input as empty, the default IQFEED level 1 field will be updated with each tick. |
| `elistener` | Function handle that specifies the function used to listen for data on the IQFEED Lookup port. |
| `ecallback` | Function handle that specifies the function that processes data event. |

## Examples

Set the data precision. Setting the connection handle property `Protocol` determines the date format for the return data based on the IQFEED version specified by the protocol.

```
q.Protocol = 5.1

q =

  iqf with properties:

        User: 'username'
```

```
    Password: 'password'
        Port: {[1x1 System.Net.Sockets.Socket]}
    PortName: {'Admin'}
    Protocol: 5.1000
```

Return level 1 data for security ABC using the default socket listener and event handler. Display the results in the MATLAB workspace variable `IQFeedLevelOneData`.

```
realtime(q,'ABC')
openvar('IQFeedLevelOneData')
```

Return level 1 data for security ABC using a field list and the defaults for the socket listener and event handler. Display the results in the MATLAB workspace variable `IQFeedLevelOneData`.

```
realtime(q,'ABC', ...
{'Symbol','Exchange ID','Last','Change','Incremental Volume'})
openvar('IQFeedLevelOneData')
```

## See Also

`close` | `history` | `iqf` | `marketdepth` | `timeseries`

### Topics

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24
“Writing and Running Custom Event Handler Functions” on page 1-26

### Introduced in R2012b

# timeseries

IQFEED asynchronous intraday tick data

## Syntax

```
timeseries(Q, S, daterange)
timeseries(Q, S, daterange, per, elistener, ecallback)
```

## Description

`timeseries(Q, S, daterange)` returns intraday ticks for the given date range using the default socket listener and event handler.

`timeseries(Q, S, daterange, per, elistener, ecallback)` returns intraday ticks for the given date range and defined period using an explicitly defined socket listener and event handler.

Data requests are returned asynchronously. For requests that return a large number of ticks, there may be a significant lag between the request and when the data is returned to the MATLAB workspace.

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf`. |
| `S` | `S` is a single security input specified as a character vector or string. |
| `daterange` | Either a scalar value that specifies how many periods of data to return or a date range of the form `{startdate,enddate}`. `startdate` and `enddate` can be input as MATLAB date numbers, character vectors, or strings. |
| `per` | Specifies, in seconds, the bar interval of the ticks used to aggregate ticks into intraday bars. |
| `elistener` | Function handle that specifies the function used to listen for data on the IQFEED Lookup port. |
| `ecallback` | Function handle that specifies the function that processes data event. |

**Note** The `timeseries` function uses the same port as the `history` function. These functions return data in either the `IQFeedTimeseriesData` or `IQFeedHistoryData` MATLAB workspace variable created by the first executed function. For different behavior, write an event handler function to process the returned data. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

## Examples

Return intraday ticks for a given date range and use the default socket listener and event handler. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{floor(now),now})
openvar('IQFeedTimeseriesData')
```

For data that is not aggregated, the fields returned are:

- Timestamp
- Last
- Last size
- Total volume
- Bid
- Ask
- TickID
- IQFEED reserved field
- IQFEED reserved field
- Basis for last

Basis for last is either a `C` that means the last qualified trade or `E` that means an extended trade.

Return the intraday ticks for a date range using the 24-hour military format, `per` of 60 seconds, and the default socket listener and event handler. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{'02/12/2012 09:30:00','02/12/2012 16:00:00'},60)
openvar('IQFeedTimeseriesData')
```

For aggregated data, the fields returned are:

- Timestamp
- High
- Low
- Open
- Close
- Total volume
- Period volume

Return the intraday ticks for a date range using the 12-hour time format.

```
timeseries(q,'ABC',{'02/12/2012 09:30:00 AM','02/12/2012 04:00:00 PM'},60)
openvar('IQFeedTimeseriesData')
```

Return the intraday ticks for a date range on the security `ABC` using the function handles `iqtimeseriesfeedlistener` and `iqtimeseriesfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{floor(now),now},[],@iqtimeseriesfeedlistener,@iqtimeseriesfeedeventhandler)
openvar('IQFeedTimeseriesData')
```

## Tips

- When you make multiple requests with multiple messages, this error might display: Warning: Error occurred while executing delegate callback: Message: The IAsyncResult object was not returned from the corresponding asynchronous method on this class.

To fix this, restart MATLAB.

## See Also

`close` | `history` | `iqf` | `marketdepth` | `realtime`

**Topics**

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-24
“Writing and Running Custom Event Handler Functions” on page 1-26

**Introduced in R2012b**

# factset

FactSet connection

## Description

The `factset` function creates a `factset` object. The `factset` object represents a FactSet connection.

After you create a `factset` object, you can use the object functions to retrieve current and historical data for securities. Retrieve data for various fields within formula libraries. For credentials, contact FactSet Research Systems.

## Creation

### Syntax

```
c = factset(username,serialnumber,password,customerid)
```

#### Description

`c = factset(username,serialnumber,password,customerid)` creates a FactSet connection using a user name, serial number, password, and customer identifier.

#### Input Arguments

**`username` — FactSet user name**
character vector | string scalar

FactSet user name, specified as a character vector or string scalar. To find your user name, contact FactSet Research Systems.

Example: `'ABCD_EFGH_IJKL'`

Data Types: `char` | `string`

**`serialnumber` — FactSet serial number**
character vector | string scalar

FactSet serial number, specified as a character vector or string scalar. To find the serial number, contact FactSet Research Systems.

Example: `'ABCDEFG'`

Data Types: `char` | `string`

**`password` — FactSet password**
character vector | string scalar

FactSet password, specified as a character vector or string scalar. To find your password, contact FactSet Research Systems.

Example: `'XXXXXXXX'`

Data Types: `char` | `string`

**`customerid` — Customer identifier**
character vector | string scalar

Customer identifier, specified as a character vector or string scalar. To find your customer identifier, contact FactSet Research Systems.

Example: `'CUSTOMERID1'`

Data Types: `char` | `string`

## Properties

**`user` — FactSet user name**
character vector

FactSet user name, specified as a character vector. For details, contact FactSet Research Systems.

The `factset` function sets this property using the `username` input argument.

Example: `'ABCD_EFGH_IJKL'`

Data Types: `char`

**`serial` — FactSet serial number**
character vector

FactSet serial number, specified as a character vector. For details, contact FactSet Research Systems.

The `factset` function sets this property using the `serialnumber` input argument.

Example: `'ABCDEFG'`

Data Types: `char`

**`password` — FactSet password**
character vector

FactSet password, specified as a character vector. For details, contact FactSet Research Systems.

The `factset` function sets this property using the `password` input argument.

Example: `'XXXXXXXX'`

Data Types: `char`

**`cid` — Customer identifier**
character vector

Customer identifier, specified as a character vector. For details, contact FactSet Research Systems.

The `factset` function sets this property using the `customerid` input argument.

Example: `'CUSTOMERID1'`

Data Types: `char`

## Object Functions

| | |
|---|---|
| fetch | Request data from FactSet |
| get | Retrieve properties of FactSet connection object |
| isconnection | Determine if connections to FactSet are valid |
| close | Close connection to FactSet |

## Examples

### Create FactSet Connection

Create a FactSet connection. Then, retrieve current data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to FactSet using a user name, serial number, password, and customer identifier. `c` is a `factset` object.

```
username = 'ABCD_EFGH_IJKL';
serialnumber = 'ABCDEFG';
password = 'XXXXXXXX';
customerid = 'CUSTOMERID1'

c = factset(username,serialnumber,password,customerid)
```

```
c =

  factset with properties:

       user: 'ABCD_EFGH_IJKL'
     serial: 'ABCDEFG'
   password: 'XXXXXXXX'
        cid: 'CUSTOMERID1'
```

Retrieve the last price of the IBM security using the FactSet connection. Use a string to specify the field name.

```
s = 'IBM';
f = "price";
d = fetch(c,s,f)
```

```
d =

  struct with fields:

      Id: {'IBM'}
    Date: 736890.00
   price: 153.63
```

`d` is a structure that contains the current data. The fields are:

- `Id` — Security name
- `Date` — MATLAB date number
- `price` — Last price

Close the FactSet connection.

```
close(c)
```

## See Also

**External Websites**
FactSet Research Systems

**Introduced before R2006a**

# close

Close connection to FactSet

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FactSet connection object created with `factset`. |

## Description

`close(Connect)` closes the connection to FactSet data.

## See Also

`factset`

**Introduced before R2006a**

# fetch

Request data from FactSet

## Syntax

```
data = fetch(Connect)
data = fetch(Connect, 'Library')
data = fetch(Connect, 'Security', 'Fields')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
'ToDate')
data = fetch(Connect, 'Security', 'FromDate',
'ToDate', 'Period')
```

## Arguments

| | |
|---|---|
| `Connect` | FactSet connection object created with the `factset` function. |
| `Library` | FactSet formula library. |
| `Security` | A MATLAB character vector, string, cell array of character vectors, or string array containing the names of securities in a format recognizable by the FactSet server. |
| `Fields` | A MATLAB character vector, string, cell array of character vectors, or string array indicating the data fields for which to retrieve data. |
| `Date` | Date character vector, string, or serial date number indicating date for the requested data. If you enter today's date, `fetch` returns yesterday's data. |
| `FromDate` | Beginning date for date range.<br><br>**Note** You can specify dates in any of the formats supported by `datestr` and `datenum` that display a year, month, and day. |
| `ToDate` | End date for date range. |

| | |
|---|---|
| `Period` | Period within date range. `Period` values are:<br>• `'d'`: daily values<br>• `'b'`: business day daily values<br>• `'m'`: monthly values<br>• `'mb'`: beginning monthly values<br>• `'me'`: ending monthly values<br>• `'q'`: quarterly values<br>• `'qb'`: beginning quarterly values<br>• `'qe'`: ending quarterly values<br>• `'y'`: annual values<br>• `'yb'`: beginning annual values<br>• `'ye'`: ending annual values |

## Description

`data = fetch(Connect)` returns the names of all available formula libraries.

`data = fetch(Connect, 'Library')` returns the valid field names for a given formula library.

`data = fetch(Connect, 'Security', 'Fields')` returns data for the specified security and fields.

`data = fetch(Connect, 'Security', 'Fields', 'Date')` returns security data for the specified fields on the requested date.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')` returns security data for the specified fields for the date range `FromDate` to `ToDate`.

`data = fetch(Connect, 'Security', 'FromDate', 'ToDate', 'Period')` returns security data for the date range `FromDate` to `ToDate` with the specified period.

## Examples

### Retrieving Names of Available Formula Libraries

Obtain the names of available formula libraries:

```
D = fetch(Connect)
```

### Retrieving Valid Field Names of a Specified Library

Obtain valid field names of the `FactSetSecurityCalcs` library:

```
D = fetch(Connect, 'fs')
```

### Retrieving the Closing Price of a Specified Security

Obtain the closing price of the security `IBM`:

```
D = fetch(Connect, 'IBM', 'price')
```

#### Retrieving the Closing Price of a Specified Security Using Default Date Period

Obtain the closing price for `IBM` using the default period of the data:

```
D = fetch(C, 'IBM', 'price', '09/01/07', '09/10/07')
```

#### Retrieving the Monthly Closing Prices of a Specified Security for a Given Date Range

Obtain the monthly closing prices for `IBM` from 09/01/05 to 09/10/07:

```
D = fetch(C, 'IBM', 'price', '09/01/05', '09/10/07', 'm')
```

## See Also

`close` | `factset` | `isconnection`

#### Introduced before R2006a

# get

Retrieve properties of FactSet connection object

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FactSet connection object created with the `factset` function. |
| `PropertyName` | (Optional) A MATLAB character vector, string, cell array of character vectors, or string array containing property names. Property names are:<br>• `user`<br>• `serial`<br>• `password`<br>• `cid` |

## Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the FactSet connection object.

`value = get(Connect)` returns a MATLAB structure where each field name is the name of a property of `Connect`, and each field contains the value of that property.

## Examples

Establish a connection to FactSet data:

```
Connect = factset('Fast_User','1234','Fast_Pass','userid')
```

Retrieve the connection property value:

```
h = get(Connect)
h=
       user: 'Fast_User'
     serial: '1234'
   password: 'Fast_Pass'
        cid: 'userid'
```

Retrieve the value of the connection's `user` property:

```
get(Connect, 'user')
ans =
Fast_User
```

## See Also

`close` | `factset` | `fetch` | `isconnection`

**Introduced before R2006a**

# isconnection

Determine if connections to FactSet are valid

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FactSet connection object created with `factset`. |

## Description

`x = isconnection(Connect)` returns `x = 1` if the connection to the FactSet is valid, and `x = 0` otherwise.

## Examples

Establish a connection, `c`, to FactSet data:

```
c = factset
```

Verify that `c` is a valid connection:

```
x = isconnection(c);
x =
    1
```

## See Also

`close` | `factset` | `fetch` | `get`

**Introduced before R2006a**

# fds

FactSet Workstation connection

## Description

The `fds` function creates an `fds` object. The `fds` object represents a FactSet Workstation connection.

After you create an `fds` object, you can use the object functions to retrieve real-time data for securities. For credentials, contact FactSet Research Systems.

## Creation

### Syntax

```
c = fds(username,password)
c = fds(username,password,finfo)
```

#### Description

`c = fds(username,password)` creates a FactSet Workstation connection using a user name and password. By default, this syntax uses the field information file `rt_fields.xml`, which is found on the MATLAB path.

`c = fds(username,password,finfo)` creates a connection using the specified field information file.

#### Input Arguments

**`username` — FactSet user name**
character vector | string scalar

FactSet user name, specified as a character vector or string scalar. To find your user name, contact FactSet Research Systems.

Example: `'ABCD_EFGH_IJKL'`

Data Types: `char` | `string`

**`password` — FactSet password**
character vector | string scalar

FactSet password, specified as a character vector or string scalar. To find your password, contact FactSet Research Systems.

Example: `'XXXXXXXX'`

Data Types: `char` | `string`

**`finfo` — Field information file**
character vector | string scalar

Field information file, specified as a character vector or string scalar. To obtain the field information file, contact FactSet Research Systems. Specify the full file path to the field information file.

Example: `'C:\Program Files (x86)\FactSet\FactSetDataFeed\fdsrt-2\etc\rt_fields.xml'`

Data Types: `char` | `string`

## Properties

### `Handle` — FactSet handle

handle object

FactSet handle, specified as a handle object.

Example: `[1×1 COM.FDSRTCom_FDF]`

## Object Functions

| | |
|---|---|
| realtime | Obtain real-time data from FactSet Workstation |
| stop | Cancel real-time request |
| close | Disconnect from FactSet Workstation |

## Examples

### Create FactSet Workstation Connection

Create a FactSet Workstation connection. Then, retrieve real-time data for a security.

Connect to the FactSet Workstation using a user name and password. By default, the `fds` function uses the field information file `rt_fields.xml`, which is found on the MATLAB path. `c` is an `fds` object.

```
username = 'ABCD_EFGH_IJKL';
password = 'XXXXXXXX';

c = fds(username,password)
```

```
c =

  fds with properties:

    Handle: [1×1 COM.FDSRTCom_FDF]
```

Retrieve real-time data for the `FDS1` service and `ABCD-USA` security by using the FactSet Workstation connection. Use the default event handler function `myMessageEventHandler` to process real-time data events from the FactSet Workstation. To access the code for the default event handler function, enter `edit myMessageEventHandler` at the command line. You can write a custom function to process real-time data events differently. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

```
Srv = 'FDS1';
Sec = 'ABCD-USA';
Cb = @(varargin)myMessageEventHandler(varargin);
t = realtime(c,Srv,Sec,Cb)
```

```
t =

     1

ABCD-USA:D 11-Sep-2017 14:04:53 6.27
ABCD-USA:D 11-Sep-2017 14:07:00 6.29
...
```

The `realtime` function returns a data tag `t` for the real-time request. Then, the event handler function returns the following data to the Command Window:

- Security name
- Date
- Time
- Last price

Stop real-time data retrieval.

```
stop(c,t)
```

Close the FactSet Workstation connection.

```
close(c)
```

### Create FactSet Workstation Connection Using Field Information File

Create a FactSet Workstation connection and specify the field information file. Then, retrieve real-time data for a security.

Connect to the FactSet Workstation using a user name, password, and field information file. `c` is an `fds` object.

```
username = 'ABCD_EFGH_IJKL';
password = 'XXXXXXXX';
finfo = 'C:\Program Files (x86)\FactSet\FactSetDataFeed\fdsrt-2\etc\rt_fields.xml';

c = fds(username,password,finfo)

c =

  fds with properties:

    Handle: [1×1 COM.FDSRTCom_FDF]
```

Retrieve real-time data for the `FDS1` service and `ABCD-USA` security by using the FactSet Workstation connection. Use the default event handler function `myMessageEventHandler` to process real-time data events from the FactSet Workstation. To access the code for the default event handler function, enter `edit myMessageEventHandler` at the command line. You can write a custom function to process real-time data events differently. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

```
Srv = 'FDS1';
Sec = 'ABCD-USA';
Cb = @(varargin)myMessageEventHandler(varargin);
t = realtime(c,Srv,Sec,Cb)
```

```
t =

     1

ABCD-USA:D 11-Sep-2017 14:04:53 6.27
ABCD-USA:D 11-Sep-2017 14:07:00 6.29
...
```

The `realtime` function returns a data tag `t` for the real-time request. Then, the event handler function returns the following data to the Command Window:

- Security name
- Date
- Time
- Last price

Stop real-time data retrieval.

```
stop(c,t)
```

Close the FactSet Workstation connection.

```
close(c)
```

## See Also

`factset`

**Topics**

“Writing and Running Custom Event Handler Functions” on page 1-26

**External Websites**

FactSet Research Systems

**Introduced in R2013a**

# realtime

Obtain real-time data from FactSet Workstation

## Syntax

```
T = realtime(c,Srv,Sec,Cb)
T = realtime(c,Srv,Sec)
```

## Description

`T = realtime(c,Srv,Sec,Cb)` asynchronously requests real-time or streaming data from the FactSet Workstation.

`T = realtime(c,Srv,Sec)` asynchronously requests real-time or streaming data from the FactSet Workstation. When `Cb` is not specified, the default message event handler `factsetMessageEventHandler` is used.

## Examples

### Request FactSet Workstation Real-Time Data with User-Defined Event Handler

To request real-time or streaming data for the symbol `'ABDC-USA'` from the service `'FDS1'`, a user-defined event handler (`myMessageEventHandler`) is used to process message events using this syntax.

```
t = realtime(c,'FDS1','ABCD-USA',@(varargin)myMessageEventHandler(varargin))
```

### Request FactSet Workstation Real-Time Data Using Default Event Handler

To request real-time or streaming data for the symbol `'ABDC-USA'` from the service `'FDS1'`, using this syntax.

```
t = realtime(c,'FDS1','ABCD-USA')
```

The default event handler is used which returns a structure `X` to the base MATLAB workspace containing the latest data for the symbol `'ABCD-USA'`. `X` is updated as new message events are received.

## Input Arguments

### `c` — FactSet Workstation connection

connection object

FactSet Workstation connection, specified as a connection object created using `fds`.

### `Srv` — Data source or supplier

character vector | string scalar

Data source or supplier, specified as a character vector or string scalar.

Example: `'FDS1'`

Data Types: `char` | `string`

**Sec — Security symbol**
character vector | string scalar

Security symbol, specified as a character vector or string scalar.

Example: `'ABCD-USA'`

Data Types: `char` | `string`

**`Cb` — Event handler**
function handle

Event handler, specified as a function handle requests real-time or streaming data from the service FactSet Workstation.

If `Cb` is not specified, the default message event handler `factsetMessageEventHandler` is used.

Example: `@(varargin)myMessageEventHandler(varargin)`

Data Types: `function_handle`

## Output Arguments

**`T` — Real-time data tag**
nonnegative integer

Real-time data tag, returned as a nonnegative integer from FactSet Workstation.

## See Also

`close` | `fds` | `stop`

**Topics**
“Writing and Running Custom Event Handler Functions” on page 1-26

**Introduced in R2013a**

# stop

Cancel real-time request

## Syntax

```
stop(c,T)
```

## Description

`stop(c,T)` cancels a real-time request. This function cleans up resources associated with real-time requests that are no longer needed.

## Examples

### Cancel FactSet Workstation Real-Time Request

Terminate a FactSet Workstation real-time request.

```
T = realtime(c,'FDS1','GOOG-USA')
stop(c,T)
```

## Input Arguments

### c — FactSet Workstation connection

connection object

FactSet Workstation connection, specified as a connection object created using `fds`.

### T — Real-time request tag

nonnegative integer

Real-time request tag, specified using `realtime`.

Data Types: `double`

## See Also

`close` | `fds` | `realtime`

### Introduced in R2013a

# close

Disconnect from FactSet Workstation

## Syntax

```
close(c)
```

## Description

`close(c)` disconnects from the FactSet Workstation given the connection object, `F`.

## Examples

### Close FactSet Workstation Connection

Close the FactSet Workstation connection.

```
T = realtime(c,'FDS1','GOOG-USA')
close(c)
```

## Input Arguments

### c — FactSet Workstation connection

connection object

FactSet Workstation connection, specified as a connection object created using `fds`.

## See Also

`fds` | `realtime` | `stop`

### Introduced in R2013a

# fred

Connect to FRED data servers

## Description

The `fred` function creates a `fred` object. The `fred` object represents a FRED connection.

After you create a `fred` object, you can use the object functions to retrieve economic data for a FRED series. You can also retrieve data for a specific date or date range.

## Creation

### Syntax

```
c = fred
c = fred(url)
```

#### Description

`c = fred` returns a FRED connection to the FRED data server using the default URL `'https://fred.stlouisfed.org/'`.

`c = fred(url)` returns a FRED connection using the specified URL.

#### Input Arguments

**`url` — URL of FRED data server**
character vector | string scalar

URL of the FRED data server, specified as a character vector or string scalar.

Example: `'https://fred.stlouisfed.org/'`

Data Types: `char` | `string`

### Properties

**`URL` — URL of FRED data server**
character vector

URL of the FRED data server, specified as a character vector.

The `fred` function sets this property using the `url` input argument.

Example: `'https://fred.stlouisfed.org/'`

Data Types: `char`

**`IP` — IP address**
`[]` (default) | character vector

IP address of the proxy server, specified as a character vector.

Data Types: `char`

**`Port` — Port number**
`[]` (default) | numeric scalar

Port number of the proxy server, specified as a numeric scalar.

Data Types: `double`

**`DataReturnFormat` — Data return format**
`[]` (default) | `'table'` | `'timetable'`

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| `[]` (default) | `structure` |
| `'table'` | `table` |
| `'timetable'` | `timetable` |

You can specify these values using a character vector or string (for example, `"table"`).

When you create a `fred` object, the `fred` function leaves this property unset. Set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'table';
```

After setting the `DataReturnFormat` property, use the `fetch` function to retrieve data.

**`DatetimeType` — Date and time data type**
`[]` (default) | `'datetime'`

Date and time data type, specified as one of these values.

| Value | Data Type of Returned Data |
|---|---|
| `[]` (default) | MATLAB date numbers |
| `'datetime'` | `datetime` array |

You can specify these values using a character vector or string (for example, `"datetime"`).

When you create a `fred` object, the `fred` function leaves this property unset. Set this property value manually at the command line or in a script using dot notation, for example:

```
c.DatetimeType = 'datetime';
```

After setting the `DatetimeType` property, use the `fetch` function to retrieve data.

## Object Functions

| | |
|---|---|
| fetch | Request data from FRED data servers |
| isconnection | Determine if connections to FRED data servers are valid |
| close | Close connections to FRED data servers |

## Examples

### Connect to FRED

Connect to the FRED® data server, and then retrieve historical data for a series.

Connect to the FRED data server.

```
c = fred
```

`c` is a FRED connection with these properties:

- URL for the FRED data server
- IP address of the proxy server
- Port number of the proxy server
- Date and time data type for returned data
- Data return format for returned data

Retrieve the `ip` property of the FRED connection `c`.

```
c.IP
```

Retrieve the `port` property of the FRED connection `c`.

```
c.Port
```

Adjust the display data format for currency.

```
format bank
```

Retrieve all historical data for the US / Euro Foreign Exchange Rate series. `d` contains the series description.

```
series = 'DEXUSEU';
```

```
d = fetch(c,series);
```

Close the FRED connection.

```
close(c)
```

### Connect to FRED with URL

Connect to the FRED® data server using a URL, and then retrieve historical data for a series.

Connect to the FRED data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url)
```

`c` is a FRED connection with these properties:

- URL for the FRED data server
- IP address of the proxy server
- Port number of the proxy server
- Date and time data type for returned data
- Data return format for returned data

Retrieve the `ip` property of the FRED connection `c`.

```
c.IP
```

Retrieve the `port` property of the FRED connection `c`.

```
c.Port
```

Adjust the display data format for currency.

```
format bank
```

Retrieve all historical data for the US / Euro Foreign Exchange Rate series. `d` contains the series description.

```
series = 'DEXUSEU';

d = fetch(c,series);
```

Close the FRED connection.

```
close(c)
```

## See Also

**Topics**
“Retrieve Historical Data Using FRED” on page 1-19

**Introduced in R2006b**

# close

Close connections to FRED data servers

## Syntax

```
close(c)
```

## Arguments

| | |
|---|---|
| c | FRED connection object created with `fred`. |

## Description

`close(c)` closes the connection to the FRED data server.

## Examples

### Close FRED® Connection

Connect to the data server at the URL `https://research.stlouisfed.org/fred2/`.

```
c = fred('https://fred.stlouisfed.org/');
```

Adjust the display data format for currency.

```
format bank
```

Retrieve all historical data for the US / Euro Foreign Exchange Rate series.

```
series = 'DEXUSEU';

d = fetch(c,series);
```

`d` contains the series description.

Close the FRED® connection.

```
close(c)
```

## See Also

`fred`

### Topics

“Retrieve Historical Data Using FRED” on page 1-19

### Introduced in R2006b

# fetch

Request data from FRED data servers

## Syntax

```
d = fetch(c,series)
d = fetch(c,series,date)
d = fetch(c,series,startdate,enddate)
```

## Description

`d = fetch(c,series)` returns FRED data using the FRED connection `c` and the specified FRED series.

`d = fetch(c,series,date)` returns FRED data for a specific date.

`d = fetch(c,series,startdate,enddate)` returns FRED data for the date range from `startdate` through `enddate`.

## Examples

### Fetch All Available FRED® Data

Connect to the FRED® data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

Fetch all available daily foreign exchange rates between the US dollar and the Euro using the series `'DEXUSEU'`.

```
series = 'DEXUSEU';
d = fetch(c,series)
```

```
d =

  struct with fields:

                Title: ' U.S. / Euro Foreign Exchange Rate'
             SeriesID: ' DEXUSEU'
               Source: ' Board of Governors of the Federal Reserve System (US)'
              Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
            Frequency: ' Daily'
                Units: ' U.S. Dollars to One Euro'
            DateRange: ' 1999-01-04 to 2017-02-03'
          LastUpdated: ' 2017-02-06 3:52 PM CST'
                Notes: ' Noon buying rates in New York City for cable transfers payable in forei
                 Data: [4720×2 double]
```

`d.Data` is an N-by-2 double array that contains dates in the first column and the series values in the second column.

Close the FRED® connection.

```
close(c)
```

### Fetch FRED® Data for Date

Connect to the FRED® data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

Adjust the display data format for currency.

```
format bank
```

Fetch data for a day three months ago using the series `'DTB6'`.

```
series = 'DTB6';
date = floor(now)-90;
d = fetch(c,series,date)
```

```
d =

  struct with fields:

                Title: ' 6-Month Treasury Bill: Secondary Market Rate'
             SeriesID: ' DTB6'
               Source: ' Board of Governors of the Federal Reserve System (US)'
              Release: ' H.15 Selected Interest Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
            Frequency: ' Daily'
                Units: ' Percent'
            DateRange: ' 1958-12-09 to 2017-02-06'
          LastUpdated: ' 2017-02-07 3:51 PM CST'
                Notes: ' Discount Basis'
                 Data: [736644.00 0.58]
```

`d.Data` is an N-by-2 double array that contains the date in the first column and the series value in the second column.

Close the FRED® connection.

```
close(c)
```

### Fetch FRED® Data for Date Range

Connect to the FRED® data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

Adjust the display data format for currency.

```
format bank
```

Retrieve historical data for the US / Euro Foreign Exchange Rate series.

```
series = 'DEXUSEU';
```

Fetch five months of data from January 1, 2007 through June 1, 2007.

```
startdate = '01/01/2007';
enddate = '06/01/2007';
d = fetch(c,series,startdate,enddate)
```

```
d =

  struct with fields:

                Title: ' U.S. / Euro Foreign Exchange Rate'
             SeriesID: ' DEXUSEU'
               Source: ' Board of Governors of the Federal Reserve System (US)'
              Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
            Frequency: ' Daily'
                Units: ' U.S. Dollars to One Euro'
            DateRange: ' 1999-01-04 to 2017-02-03'
          LastUpdated: ' 2017-02-06 3:52 PM CST'
                Notes: ' Noon buying rates in New York City for cable transfers payable in forei
                 Data: [110×2 double]
```

`d.Data` is an N-by-2 double array that contains dates in the first column and the series values in the second column.

Close the FRED® connection.

```
close(c)
```

### Retrieve Available FRED Data as Table

Connect to the FRED data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

Adjust the display format for currency.

```
format bank
```

Set the data return format to table using the `DataReturnFormat` property of the `fred` object.

```
c.DataReturnFormat = 'table';
```

Fetch all available daily foreign exchange rates between the US dollar and the Euro using the series `'DEXUSEU'`. The table `d` contains one row for the series. Each variable describes a piece of information about the series.

```
series = 'DEXUSEU';
d = fetch(c,series)

d=1×11 table
                  Title                       SeriesID                                Source
    ____________________________________    _________    ____________________________________

    ' U.S. / Euro Foreign Exchange Rate'    ' DEXUSEU'    ' Board of Governors of the Federal Res
```

Access the data in the series from the `Data` variable.

```
data = d.Data{1};
```

Display the first few foreign exchange rates. The first variable in the table is the date and the second variable is the foreign exchange rate. Also, the first variable is an array of MATLAB® date numbers.

```
head(data)

ans=8×2 table
      Var1       Var2
    _________    ____

    730124.00    1.18
    730125.00    1.18
    730126.00    1.16
    730127.00    1.17
    730128.00    1.16
    730131.00    1.15
    730132.00    1.15
    730133.00    1.17
```

Close the FRED connection.

```
close(c)
```

### Retrieve Available FRED Data as Table with Datetime

Connect to the FRED data server using the URL `'https://fred.stlouisfed.org/'`.

```
url = 'https://fred.stlouisfed.org/';
c = fred(url);
```

Adjust the display format for currency.

```
format bank
```

Set the data return format to table using the `DataReturnFormat` property of the `fred` object. Also, set the date and time format to `datetime` array using the `DatetimeType` property.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Fetch all available daily foreign exchange rates between the US dollar and the Euro using the series `'DEXUSEU'`. The table `d` contains one row for the series. Each variable describes a piece of information about the series.

```
series = 'DEXUSEU';
d = fetch(c,series)
```

```
d=1×11 table
                   Title                      SeriesID                              Source
    ____________________________________    __________    ____________________________________

    ' U.S. / Euro Foreign Exchange Rate'    ' DEXUSEU'    ' Board of Governors of the Federal Res
```

Access the data in the series from the `Data` variable.

```
data = d.Data{1};
```

Display the first few foreign exchange rates. The first variable in the table is the date and the second variable is the foreign exchange rate. Also, the first variable is a `datetime` array.

```
head(data)
```

```
ans=8×2 table
            Var1            Var2
    ____________________    ____

    04-Jan-1999 00:00:00    1.18
    05-Jan-1999 00:00:00    1.18
    06-Jan-1999 00:00:00    1.16
    07-Jan-1999 00:00:00    1.17
    08-Jan-1999 00:00:00    1.16
    11-Jan-1999 00:00:00    1.15
    12-Jan-1999 00:00:00    1.15
    13-Jan-1999 00:00:00    1.17
```

Close the FRED connection.

```
close(c)
```

## Input Arguments

### `c` — FRED connection

connection object

FRED connection, specified as a connection object created using `fred`.

### `series` — FRED series

character vector | string scalar

FRED series, specified as a character vector or string scalar.

Example: `'DEXUSEU'`

Data Types: `char` | `string`

**`date` — Date**
`datetime` | matrix | character vector | string scalar

Date, specified as a `datetime` value, matrix, character vector, or string scalar. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime` | `string`

**`startdate` — Start date**
`datetime` | matrix | character vector | string scalar

Start date in a date range, specified as a `datetime` value, matrix, character vector, or string scalar. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime` | `string`

**`enddate` — End date**
`datetime` | matrix | character vector | string scalar

End date in a date range, specified as a `datetime` value, matrix, character vector, or string scalar. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime` | `string`

## Output Arguments

**`d` — FRED data**
structure

FRED data, returned as a structure. For details about FRED data, see https://research.stlouisfed.org/fred2/.

## See Also
`close` | `fred` | `isconnection`

**Topics**
“Retrieve Historical Data Using FRED” on page 1-19

**Introduced in R2006b**

# get

Retrieve properties of FRED connection objects

## Syntax

```
value = get(c,'PropertyName')
value = get(c)
```

## Arguments

| | |
|---|---|
| `c` | FRED connection object created with `fred`. |
| `'PropertyName'` | A MATLAB character vector, string, cell array of character vectors, or string array containing property names. Property names are:<br>• `'url'`<br>• `'ip'`<br>• `'port'` |

## Description

`value = get(c,'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the FRED connection object.

`value = get(c)` returns the value for all properties.

## Examples

Establish a connection, `c`, to a FRED data server.

```
c = fred('https://fred.stlouisfed.org/')
```

Retrieve the port and IP address for the connection.

```
p = get(c,{'port','ip'})
p =
     port: 8194
     ip: 111.222.33.444
```

## See Also

`close` | `fetch` | `isconnection`

**Topics**

“Retrieve Historical Data Using FRED” on page 1-19

**Introduced in R2006b**

# isconnection

Determine if connections to FRED data servers are valid

## Syntax

```
x = isconnection(c)
```

## Arguments

| | |
|---|---|
| `c` | FRED connection object created with `fred`. |

## Description

`x = isconnection(c)` returns `x = 1` if a connection to the FRED data server is valid, and `x = 0` otherwise.

## Examples

### Verify FRED® Connection

Establish a connection `c` to a FRED® data server.

```
c = fred('https://fred.stlouisfed.org/');
```

Verify that `c` is a valid connection.

```
x = isconnection(c)
```

```
x =

     1
```

Adjust the display data format for currency.

```
format bank
```

Retrieve all historical data for the US / Euro Foreign Exchange Rate series.

```
series = 'DEXUSEU';

d = fetch(c,series);
```

`d` contains the series description.

Close the FRED® connection.

```
close(c)
```

## See Also

`close` | `fetch` | `fred`

**Topics**

“Retrieve Historical Data Using FRED” on page 1-19

**Introduced in R2006b**

# haver

Connect to local Haver Analytics database

# Description

The `haver` function creates a `haver` object. The `haver` object represents a Haver Analytics database connection.

After you create a `haver` object, you can use the object functions to retrieve all historical data for a variable. You can also retrieve historical data in a date range.

# Creation

## Syntax

```
c = haver(databasename)
```

### Description

`c = haver(databasename)` establishes a connection to a Haver Analytics database and sets the "DatabaseName" on page 12-0 property.

---

**Requirement:** You need both read and write permissions on the database file to establish a connection. Otherwise, this error message appears at the command line: `Unable to open specified database file`.

---

### Input Arguments

**`databasename` — Database name**
character vector | string scalar

Database name, specified as a character vector or string scalar. The database name is the full path to the Haver Analytics database file on the local machine.

Example: `'C:\haver\haverd.dat'`

Data Types: `char` | `string`

## Properties

**`DatabaseName` — Database name**
character vector

This property is read-only.

Database name, specified as a character vector. The database name is the full path to the Haver Analytics database file on the local machine.

The haver function sets this property using the databasename input argument.

Example: 'C:\haver\haverd.dat'

Data Types: char

**DataReturnFormat — Data return format**
'' (default) | 'table' | 'timetable'

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| '' (default) | structure or numeric array |
| 'table' | table |
| 'timetable' | timetable |

You can specify these values using a character vector or string (for example, "table").

When you create a haver object, the haver function leaves this property unset. Set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'table';
```

The default data type value depends on the function. The following table describes the default value for each corresponding supported function.

| Supported Function | Default Data Type |
|---|---|
| fetch | numeric array |
| info | structure |
| nextinfo | structure |

**DatetimeType — Date and time data type**
'' (default) | 'datetime'

Date and time data type, specified as one of these values.

| Value | Data Type of Returned Data |
|---|---|
| '' (default) | MATLAB date numbers or character vectors |
| 'datetime' | datetime array |

You can specify these values using a character vector or string (for example, "datetime").

When you create a `haver` object, the `haver` function leaves this property unset. Set this property value manually at the command line or in a script using dot notation, for example:

```
c.DatetimeType = 'datetime';
```

The default data type value depends on the function. The following table describes the default value for each corresponding supported function.

| Supported Function | Default Data Type |
|---|---|
| `fetch` | numeric scalar |
| `info` | character vector |
| `nextinfo` | character vector |

**Note** If you leave the DataReturnFormat property set to the default value and you retrieve data using the `fetch` function, the date and time data type remains a numeric scalar. To retrieve date and time values as a `datetime` array, set the `DataReturnFormat` property to `'timetable'`, or set the `DataReturnFormat` property to `'table'` and the `DatetimeType` property to `'datetime'`.

## Object Functions

| | |
|---|---|
| close | Close Haver Analytics database |
| fetch | Request data from Haver Analytics database |
| get | Retrieve properties from Haver Analytics connection objects |
| info | Retrieve information about Haver Analytics variables |
| isconnection | Determine if connections to Haver Analytics data servers are valid |
| nextinfo | Retrieve information about next Haver Analytics variable |

## Examples

### Connect to Haver Analytics Database

Create a Haver Analytics database connection. Then, retrieve historical data for a specified date range.

Create the Haver Analytics database connection using the local Haver Analytics database file. `c` is a `haver` object.

```
databasename = 'C:\haver\usecon.dat';
c = haver(databasename)
```

```
c =

  haver with properties:

        DatabaseName: 'C:\haver\usecon.dat'
        DatetimeType: ''
    DataReturnFormat: ''
```

Adjust the display format for currency.

```
format bank
```

Retrieve historical data from January 1, 2005 through December 31, 2005 for `'FFED'`.

```
variable = 'FFED'; % return data for FFED
startdate = '01/01/2005'; % start of date range
enddate = '12/31/2005';   % end of date range

d = fetch(c,variable,startdate,enddate);
```

Display the first three rows of data. `d` contains the numeric representation of the date in the first column and the closing value in the second column.

```
d(1:3,:)

ans =

     732315.00          2.25
     732316.00          2.25
     732317.00          2.25
```

Close the Haver Analytics connection.

```
close(c)
```

## See Also

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# aggregation

Set Haver Analytics aggregation mode

## Syntax

```
X = aggregation (C)
X = aggregation (C,V)
```

## Description

`X = aggregation (C)` returns the current aggregation mode.

`X = aggregation (C,V)` sets the current aggregation mode to `V`. The following table lists possible values for `V`.

| Value of `V` | Aggregation mode | Behavior of `aggregation` function |
|---|---|---|
| `0` | `strict` | `aggregation` does not fill in values for missing data. |
| `1` | `relaxed` | `aggregation` fills in missing data based on data available in the requested period. |
| `2` | `forced` | `aggregation` fills in missing data based on some past value. |
| `-1` | Not recognized | `aggregation` resets `V` to its last valid setting. |

## See Also

`close` | `fetch` | `haver` | `info` | `isconnection` | `nextinfo`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2008b**

# close

Close Haver Analytics database

## Syntax

```
close(c)
```

## Arguments

| | |
|---|---|
| `c` | Haver Analytics connection object created with `haver`. |

## Description

`close(c)` closes the connection to the Haver Analytics database.

## Examples

Establish a connection to a Haver Analytics database:

```
c = haver('d:\work\haver\data\haverd.dat')
```

Close the connection:

```
close(c)
```

## See Also

`haver`

### Topics

“Retrieve Historical Data Using Haver Analytics” on page 1-23

### Introduced in R2007a

# fetch

Request data from Haver Analytics database

## Syntax

```
d = fetch(c,variable)
d = fetch(c,variable,startdate,enddate)
d = fetch(c,variable,startdate,enddate,period)
```

## Description

`d = fetch(c,variable)` returns historical data for the Haver Analytics variable `s`, using the connection object `c`.

`d = fetch(c,variable,startdate,enddate)` returns historical data between the dates `startdate` and `enddate`.

`d = fetch(c,variable,startdate,enddate,period)` returns historical data in time periods specified by `period`.

## Examples

### Retrieve Variable Data

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\usecon.dat');
```

Retrieve all historical data for the Haver Analytics variable `'FFED'`. The descriptor for this variable is Federal Funds [Effective] Rate (% p.a.).

```
variable = 'FFED'; % return data for FFED

d = fetch(c,variable);
```

Display the first three rows of data.

```
d(1:3,:)

ans =

     715511.00          2.38
     715512.00          2.50
     715515.00          2.50
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Variable Data for Specified Date Range

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\usecon.dat');
```

Retrieve historical data from January 1, 2005 through December 31, 2005 for `'FFED'`.

```
variable = 'FFED'; % return data for FFED
startdate = '01/01/2005'; % start of date range
enddate = '12/31/2005';   % end of date range

d = fetch(c,variable,startdate,enddate);
```

Display the first three rows of data.

```
d(1:3,:)

ans =

     732315.00          2.25
     732316.00          2.25
     732317.00          2.25
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Quarterly Data for Specified Date Range

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\usecon.dat');
```

Retrieve the information of the Haver Analytics variable `'FFED'`. The descriptor for this variable is Federal Funds [Effective] Rate (% p.a.).

```
variable = 'FFED';

x = info(c,variable);
```

`info` returns the structure `x` containing fields describing the Haver Analytics variable.

Retrieve quarterly data. When you specify a date that is outside the date range in the variable, you might experience unexpected results. To prevent this, use the `EndDate` field for the end of the date range.

```
startdate = '06/01/2000';  % start of date range
enddate = x.EndDate;       % end of date range
period = 'q';              % quarterly data
```

```
d = fetch(c,variable,startdate,enddate,period)
```

Display the first three rows of data.

```
d(1:3,:)
```

```
ans =

     730759.00          6.52
     730851.00          6.50
     730941.00          5.61
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Variable Data as Table

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\usecon.dat');
```

Adjust the display format for currency.

```
format bank
```

Set the data return format to table using the `DataReturnFormat` property of the `haver` object.

```
c.DataReturnFormat = 'table';
```

Retrieve historical data from January 1, 2005 through December 31, 2005 for `'ABQ'` and display the results. `ABQ` provides bankruptcy filings in the US.

```
variable = 'ABQ'; % return data for ABQ
startdate = '01/01/2005'; % start of date range
enddate = '12/31/2005';   % end of date range

d = fetch(c,variable,startdate,enddate)
```

```
ans =

  4×2 table

      Time       TotalBankruptcyFilings_U_S__Units_
    _________    __________________________________

    732402.00                401149.00
    732493.00                467333.00
    732585.00                542002.00
    732677.00                667431.00
```

`d` is a table with the date in the first variable and the bankruptcy amount in the second variable. The date is an array of MATLAB date numbers.

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Variable Data as Table with Datetime

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\usecon.dat');
```

Adjust the display format for currency.

```
format bank
```

Set the data return format to table using the `DataReturnFormat` property of the `haver` object. Also, set the date and time return format to a `datetime` array using the `DatetimeType` property.

```
c.DataReturnFormat = 'table';
c.DatetimeType = 'datetime';
```

Retrieve historical data from January 1, 2005 through December 31, 2005 for `'ABQ'` and display the results. `ABQ` provides bankruptcy filings in the US.

```
variable = 'ABQ'; % return data for ABQ
startdate = '01/01/2005'; % start of date range
enddate = '12/31/2005';   % end of date range

d = fetch(c,variable,startdate,enddate)
```

```
ans =

  4×2 table

           Time            TotalBankruptcyFilings_U_S__Units_
    ____________________    __________________________________

    31-Mar-2005 00:00:00                401149.00
    30-Jun-2005 00:00:00                467333.00
    30-Sep-2005 00:00:00                542002.00
    31-Dec-2005 00:00:00                667431.00
```

`d` is a table with the date and time in the first variable and the bankruptcy amount in the second variable. The first variable is a `datetime` array.

Close the Haver Analytics connection.

```
close(c)
```

## Input Arguments

### c — Haver Analytics connection

connection object

Haver Analytics connection, specified as a connection object created using `haver`.

**`variable` — Haver Analytics variable**
character vector | string scalar | cell array of character vectors | string array

Haver Analytics variable, specified as a character vector, string scalar, cell array of character vectors, or string array to denote which historical data to retrieve.

Example: `'FFED'`

Data Types: `char` | `string` | `cell`

**`startdate` — Start date**
character vector | string scalar | MATLAB date number

Start date, specified as a character vector, string scalar, or MATLAB date number that denotes the beginning of the date range to retrieve data.

Data Types: `double` | `char` | `string`

**`enddate` — End date**
character vector | string scalar | MATLAB date number

End date, specified as a character vector, string scalar, or MATLAB date number that denotes the end of the date range to retrieve data.

Data Types: `double` | `char` | `string`

**`period` — Period**
`'d'` | `'w'` | `'m'` | `'q'` | `'a'`

Period, specified as one of these values that denotes the time period for the historical data:

- `'d'` for daily values
- `'w'` for weekly values
- `'m'` for monthly values
- `'q'` for quarterly values
- `'a'` for annual values

## Output Arguments

**`d` — Historical data**
matrix

Historical data, returned as a matrix with the numeric representation of the date in the first column and the value in the second column.

## See Also
`close` | `get` | `haver` | `info` | `isconnection` | `nextinfo`

**Topics**
“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# get

Retrieve properties from Haver Analytics connection objects

## Syntax

```
V = get(c,'PropertyName')
V = get(c)
```

## Arguments

| | |
|---|---|
| `c` | Haver Analytics connection object created with `haver`. |
| `'PropertyName'` | A MATLAB character vector, string, cell array of character vectors, or string array containing property names. The property name is `Databasename`. |

## Description

`V = get(c,'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the Haver Analytics connection object.

`V = get(c)` returns a MATLAB structure, where each field name is the name of a property of `c`. Each field contains the value of the property.

## Examples

Establish a Haver Analytics connection.

```
c = haver('d:\work\haver\data\haverd.dat')
```

Retrieve the name of the Haver Analytics database.

```
V = get(c,'DatabaseName')

  V =

    DatabaseName: 'd:\work\haver\data\haverd.dat'
```

## See Also

`close` | `fetch` | `haver` | `isconnection`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# info

Retrieve information about Haver Analytics variables

## Syntax

```
D = info(c,s)
```

## Arguments

| | |
|---|---|
| `c` | Haver Analytics connection object created with `haver`. |
| `s` | Haver Analytics variable. |

## Description

`D = info(c,s)` returns information about the Haver Analytics variable, `s`.

## Examples

### Retrieve Information About Variable

Using a Haver Analytics connection, retrieve information about a variable.

Establish a Haver Analytics connection `c`.

```
c = haver('d:\work\haver\data\haverd.dat');
```

Request information for the variable `'FFED2'`. The variable `d` is a structure with a field for each piece of information.

```
s = 'FFED2';
d = info(c,s)
```

```
d =

  struct with fields:

     VarName: 'FFED2'
   StartDate: '01-Jan-1991'
     EndDate: '31-Dec-1998'
   NumberObs: 2088
   Frequency: 'D'
 DateTimeMod: '02-Apr-2007 20:46:37'
   Magnitude: 0
DecPrecision: 2
     DifType: 1
     AggType: 'AVG'
    DataType: '%'
       Group: 'Z05'
      Source: 'FRB'
```

```
  Descriptor: 'Federal Funds [Effective] Rate (% p.a.)'
 ShortSource: 'History'
  LongSource: 'Historical Series'
```

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Information About Variable as Table

Using a Haver Analytics connection, retrieve information about a variable. Return information as a table.

Establish a Haver Analytics connection `c`.

```
c = haver('d:\work\haver\data\haverd.dat');
```

Set the data return format to a table using the `DataReturnFormat` property of the `haver` object.

```
c.DataReturnFormat = 'table';
```

Request information for the variable `'ABQ'`. The variable `ABQ` provides bankruptcy filings in the US. `d` is a table with a variable for each piece of information.

```
s = 'ABQ';
d = info(c,s)
```

```
d =

  1×16 table

    VarName      StartDate         EndDate       NumberObs    Frequency          DateTimeMod
    _______    _____________    _____________    _________    _________    ____________________

     'ABQ'     '01-Jan-1980'    '01-Jan-2015'     141.00         'Q'       '28-Apr-2015 16:21:22
```

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Information About Variable with Datetime

Using a Haver Analytics connection, retrieve information about a variable. Return dates as `datetime` arrays.

Establish a Haver Analytics connection `c`.

```
c = haver('d:\work\haver\data\haverd.dat');
```

Set the date and time format to a `datetime` array using the `DatetimeType` property of the `haver` object.

```
c.DatetimeType = 'datetime';
```

Request information for the variable `'ABQ'`. The variable `ABQ` provides bankruptcy filings in the US.

```
s = 'ABQ';
d = info(c,s)
```

```
d =

  struct with fields:

         VarName: 'ABQ'
       StartDate: 01-Jan-1980
         EndDate: 01-Jan-2015
       NumberObs: 141.00
       Frequency: 'Q'
     DateTimeMod: 28-Apr-2015 16:21:22
       Magnitude: 0
    DecPrecision: 0
         DifType: 0
         AggType: 'SUM'
        DataType: 'Units'
           Group: 'C13'
          Source: 'USC'
      Descriptor: 'Total Bankruptcy Filings, U.S. (Units)'
     ShortSource: 'USCOURTS'
      LongSource: 'Administrative Office of the U.S. Courts'
```

`d` is a structure with a field for each piece of information. The dates are `datetime` arrays.

Close the Haver Analytics connection.

```
close(c)
```

## See Also

`close` | `get` | `haver` | `isconnection` | `nextinfo`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# isconnection

Determine if connections to Haver Analytics data servers are valid

## Syntax

```
X = isconnection(c)
```

## Arguments

| | |
|---|---|
| `c` | Haver Analytics connection object created with `haver`. |

## Description

`X = isconnection(c)` returns `X = 1` if the connection is a valid Haver Analytics connection, and `X = 0` otherwise.

## Examples

Establish a Haver Analytics connection `c`:

```
c = HAVER('d:\work\haver\data\haverd.dat');
```

Verify that `c` is a valid Haver Analytics connection:

```
X = isconnection(c)
X = 1
```

## See Also

`close` | `fetch` | `get` | `haver`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# nextinfo

Retrieve information about next Haver Analytics variable

## Syntax

```
D = nextinfo(c,s)
```

## Arguments

| | |
|---|---|
| c | Haver Analytics connection object created with the haver function. |
| s | Haver Analytics variable. |

## Description

`D = nextinfo(c,s)` returns information for the next Haver Analytics variable after the variable, `s`.

## Examples

### Retrieve Information About Next Variable

Establish a Haver Analytics connection.

```
c = haver('d:\work\haver\data\usecon.dat');
```

Request information for the variable following `'FFED'`. The variable `d` is a structure with a field for each piece of information.

```
s = 'FFED';
d = nextinfo(c,s)
```

```
d =

  struct with fields:

     VarName: 'FFED2'
   StartDate: '01-Jan-1991'
     EndDate: '31-Dec-1998'
   NumberObs: 2088
   Frequency: 'D'
 DateTimeMod: '02-Apr-2007 20:46:37'
   Magnitude: 0
DecPrecision: 2
     DifType: 1
     AggType: 'AVG'
    DataType: '%'
       Group: 'Z05'
      Source: 'FRB'
  Descriptor: 'Federal Funds [Effective] Rate (% p.a.)'
 ShortSource: 'History'
  LongSource: 'Historical Series'
```

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Information About Next Variable as Table

Establish a Haver Analytics connection.

```
c = haver('d:\work\haver\data\usecon.dat');
```

Set the data return format to a table using the `DataReturnFormat` property of the `haver` object.

```
c.DataReturnFormat = 'table';
```

Request information for the variable following `'ABQ'`. The variable `d` is a table with a variable for each piece of information.

```
s = 'ABQ';
d = nextinfo(c,s)
```

```
d =

  1×16 table

    VarName       StartDate         EndDate       NumberObs    Frequency         DateTimeMod
    ________    _____________    _____________    _________    _________    ____________________

    'ACPIF1'    '01-Jan-1967'    '01-Apr-2015'     580.00         'M'       '22-May-2015 14:38:31
```

Close the Haver Analytics connection.

```
close(c)
```

### Retrieve Information About Next Variable with Datetime

Establish a Haver Analytics connection.

```
c = haver('d:\work\haver\data\usecon.dat');
```

Set the date and time format to a `datetime` array using the `DatetimeType` property of the `haver` object.

```
c.DatetimeType = 'datetime';
```

Request information for the variable following `'ABQ'`.

```
s = 'ABQ';
d = nextinfo(c,s)
```

```
d =

  struct with fields:

          VarName: 'ACPIF1'
        StartDate: 01-Jan-1967
```

```
         EndDate: 01-Apr-2015
       NumberObs: 580.00
       Frequency: 'M'
    DateTimeMod: 22-May-2015 14:38:33
       Magnitude: 0
    DecPrecision: 2.00
        DifType: 1.00
        AggType: 'NA'
       DataType: '%'
          Group: 'P25'
         Source: 'STLF'
     Descriptor: 'Atlanta Fed Flexible CPI (SAAR, %chg)'
    ShortSource: 'FRBATL'
     LongSource: 'Federal Reserve Bank of Atlanta'
```

The variable `d` is a structure with a field for each piece of information. The dates are `datetime` arrays.

Close the Haver Analytics connection.

```
close(c)
```

## See Also

`close` | `get` | `haver` | `info` | `isconnection`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# havertool

Run Haver Analytics graphical user interface (GUI)

## Syntax

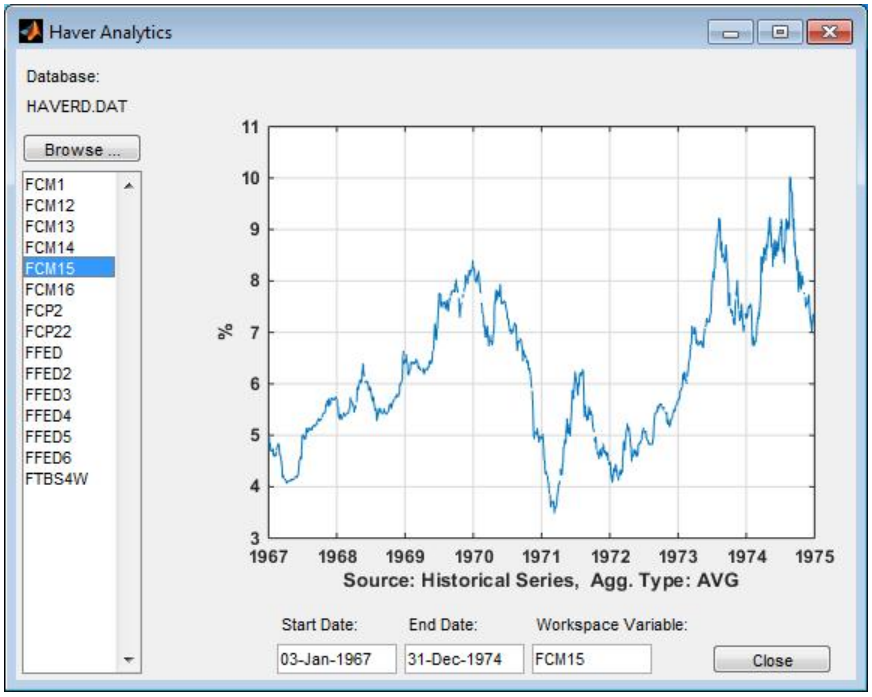
```
havertool(c)
```

## Arguments

| | |
|---|---|
| c | Haver Analytics connection object created with `haver`. |

## Description

`havertool(c)` runs the Haver Analytics graphical user interface (GUI). The GUI appears in the following figure.




The GUI fields and buttons are:

- **Database**: The currently selected Haver Analytics database.
- **Browse**: Allows you to browse for Haver Analytics databases, and populates the variable list with the variables in the database you specify.
- **Start Date**: The data start date of the selected variable.
- **End Date**: The data end date of the selected variable.
- **Workspace Variable**: The MATLAB variable to which `havertool` writes data for the currently selected Haver Analytics variable.

- **Close**: Closes all current connections and the Haver Analytics GUI.

## Examples

Establish a Haver Analytics connection `H`:

```
c = haver('d:\work\haver\data\haverd.dat');
```

Open the graphical user interface (GUI) demonstration:

```
havertool(c)
```

## See Also

`haver`

**Topics**

“Retrieve Historical Data Using Haver Analytics” on page 1-23

**Introduced in R2007a**

# kx

Connect to Kx Systems, Inc. kdb+ databases

## Description

The kx function creates a kx object. The kx object represents a Kx Systems, Inc. kdb+ database connection.

After you create a kx object, you can use the object functions to run Kx Systems, Inc. kdb+ commands, retrieve data from the Kx Systems, Inc. kdb+ database, and write data back to the database.

Before you connect to the database, add the Kx Systems, Inc. file `jdbc.jar` to the MATLAB Java class path by using the `javaaddpath` command. The following code adds the JAR file to the MATLAB Java class path. This code assumes that the full path of the JAR file is `c:\q\java\jdbc.jar`.

```
javaaddpath c:\q\java\jdbc.jar
```

Alternatively, add the JAR file to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

---

**Note** Earlier versions of the Kx Systems, Inc. kdb+ database provide this JAR file with the name `kx.jar`. If you are running an earlier version of the database, rename `kx.jar` as `jdbc.jar` to add this file to the MATLAB Java class path.

---

## Creation

### Syntax

```
c = kx(ipaddress,port)
c = kx(ipaddress,port,customerid)
```

#### Description

`c = kx(ipaddress,port)` connects to a Kx Systems, Inc. kdb+ database and sets the ipaddress and port properties.

`c = kx(ipaddress,port,customerid)` uses a customer identifier for the database connection.

#### Input Arguments

**`customerid` — Customer identifier**
character vector | string scalar

Customer identifier, specified as a character vector or string scalar. The customer identifier consists of a user name and password separated by a colon, such as `'username:password'`.

Data Types: `char` | `string`

## Properties

### `handle` — Handle

`[]` (default) | handle object

Handle, specified as a Kx Systems, Inc. kdb+ database handle object. For details, contact Kx Systems, Inc.

Example: `[1×1 c]`

Data Types: `double`

### `ipaddress` — IP address

`[]` (default) | character vector | string scalar

IP address of the machine where the kdb+ database is located, specified as a character vector or string scalar.

Example: `'localhost'`

Data Types: `char` | `string`

### `port` — Port number

`[]` (default) | numeric scalar

Port number of the machine where the kdb+ database is located, specified as a numeric scalar.

Example: `5001`

Data Types: `double`

## Object Functions

| | |
|---|---|
| isconnection | Determine if connections to Kx Systems, Inc. kdb+ databases are valid |
| close | Close connections to Kx Systems, Inc. kdb+ databases |
| get | Retrieve Kx Systems, Inc. kdb+ connection object properties |
| exec | Run Kx Systems, Inc. kdb+ commands |
| fetch | Request data from Kx Systems, Inc. kdb+ databases |
| tables | Retrieve table names from Kx Systems, Inc. kdb+ databases |
| insert | Write data to Kx Systems, Inc. kdb+ databases |

## Examples

### Connect to Kx Systems, Inc. kdb+ Database

Create a kdb+ database connection. Then, retrieve data from the database.

Run this command at the DOS command prompt.

```
q tradedata.q -p 5001
```

Connect to a kdb+ database using an IP address and port number. `c` is a `kx` object.

```
ipaddress = 'localhost';
port = 5001;
c = kx(ipaddress,port)
```

```
c =

  kx with properties:

       handle: [1×1 c]
    ipaddress: 'localhost'
         port: 5001
```

Retrieve data from the kdb+ database.

```
ksql = 'select from trade';
d = fetch(c,ksql)
```

```
d =
          sec: {5000x1 cell}
        price: [5000x1 double]
       volume: [5000x1 int32]
     exchange: [5000x1 double]
         date: [5000x1 double]
```

d is a structure that contains these fields:

- Security
- Price
- Volume
- Exchange
- Date

Close the kdb+ database connection.

```
close(c)
```

### Connect to Kx Systems, Inc. kdb+ Database Using Customer Identifier

Create a kdb+ database connection using the customer identifier. Then, retrieve data from the database.

Run this command at the DOS command prompt.

```
q tradedata.q -p 5001
```

Connect to a kdb+ database using an IP address, port number, and customer identifier. The customer identifier consists of a character vector that contains the user name and password separated by a colon. c is a kx object.

```
ipaddress = 'localhost';
port = 5001;
customerid = 'username:password';
c = kx(ipaddress,port,customerid)
```

```
c =

  kx with properties:

       handle: [1×1 c]
```

```
    ipaddress: 'localhost'
         port: 5001
```

Retrieve data from the kdb+ database.

```
ksql = 'select from trade';
d = fetch(c,ksql)
```

```
d =
          sec: {5000x1 cell}
        price: [5000x1 double]
       volume: [5000x1 int32]
     exchange: [5000x1 double]
         date: [5000x1 double]
```

d is a structure that contains these fields:

- Security
- Price
- Volume
- Exchange
- Date

Close the kdb+ database connection.

```
close(c)
```

## See Also

**External Websites**
Kx Systems, Inc. Documentation
MATLAB client for kdb+

**Introduced in R2007a**

# close

Close connections to Kx Systems, Inc. kdb+ databases

## Syntax

```
close(k)
```

## Arguments

| | |
|---|---|
| k | Kx Systems, Inc. kdb+ connection object created with kx. |

## Description

`close(k)` closes the connection to the Kx Systems, Inc. kdb+ database.

## Examples

Close the connection, k, to the Kx Systems, Inc. kdb+ database:

```
close(k)
```

## See Also

kx

**Introduced in R2007a**

# exec

Run Kx Systems, Inc. kdb+ commands

## Syntax

```
exec(k,command)
exec(k,command,p1,p2,p3)
exec(k,command,p1)
exec(k,command,p1,p2)
exec(k,command,p1,p2,p3)
exec(k,command,p1,p2,p3,sync)
```

## Arguments

| | |
|---|---|
| k | Kx Systems, Inc. kdb+ connection object created with kx. |
| command | Kx Systems, Inc. kdb+ command issued using the Kx Systems, Inc. kdb+ connection object created with the kx function. |
| p1,p2,p3 | Input parameters for Command. |

## Description

`exec(k,command)` executes the specified `command` in Kx Systems, Inc. kdb+ without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the specified `command` with one or more input parameters without waiting for a response.

`exec(k,command,p1)` executes the given `command` with one input parameter without waiting for a response.

`exec(k,command,p1,p2)` executes the given `command` with two input parameters without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the given `command` with three input parameters without waiting for a response.

`exec(k,command,p1,p2,p3,sync)` executes the given `command` with three input parameters synchronously and waits for a response from the database. Enter unused parameters as empty. You can enter `sync` as 0 (default) for asynchronous commands and as 1 for synchronous commands.

## Examples

Retrieve the data in the table `trade` using the connection to the Kx Systems, Inc. kdb+ database, K:

```
k = kx('localhost',5001);
```

Use the `exec` command to sort the data in the table `trade` in ascending order.

```
exec(k,'`date xasc`trade');
```

Subsequent data requests also sort returned data in ascending order.

After running

```
q tradedata.q -p 5001
```

at the DOS prompt, the commands

```
k = kx('localhost',5001);
exec(k,'`DATE XASC `TRADE');
```

sort the data in the table `trade` in ascending order. Data later fetched from the table will be ordered in this manner.

## See Also

`fetch` | `insert` | `kx`

**External Websites**

Kx Systems, Inc. Documentation
MATLAB client for kdb+

**Introduced in R2007a**

# fetch

Request data from Kx Systems, Inc. kdb+ databases

## Syntax

```
d = fetch(k,ksql)
d = fetch(k,ksql,p1)
d = fetch(k,ksql,p1,p2)
d = fetch(k,ksql,p1,p2,p3)
```

## Arguments

| | |
|---|---|
| `k` | Kx Systems, Inc. kdb+ connection object created with `kx`. |
| `ksql` | The Kx Systems, Inc. kdb+ command. |
| `p1,p2,p3` | Input parameters for the `ksql` command. |

## Description

`d = fetch(k,ksql)` returns data from a Kx Systems, Inc. kdb+ database in a MATLAB structure where `k` is the Kx Systems, Inc. kdb+ object and `ksql` is the Kx Systems, Inc. kdb+ command. `ksql` can be any valid kdb+ command. The output of the `fetch` function is any data resulting from the command specified in `ksql`.

`d = fetch(k,ksql,p1)` executes the command specified in `ksql` with one input parameter, and returns the data from this command.

`d = fetch(k,ksql,p1,p2)` executes the command with two input parameters.

`d = fetch(k,ksql,p1,p2,p3)` executes the command with three input parameters.

## Examples

Run the following command from a DOS prompt to specify the port number `5001`:

```
q tradedata.q -p 5001
```

Connect to a Kx Systems, Inc. server using IP address `'localhost'` and port number `5001`:

```
k = kx('localhost',5001);
```

Retrieve data using the command `'select from trade'`:

```
d = fetch(k,'select from trade');
d =
      sec: {5000x1 cell}
    price: [5000x1 double]
   volume: [5000x1 int32]
 exchange: [5000x1 double]
     date: [5000x1 double]
```

Retrieve data, passing an input parameter `'ACME'` to the command `'totalvolume'`:

```
d = fetch(k,'totalvolume','ACME');
d =
      volume: [1253x1 int32]
```

This is the total trading volume for the security `ACME` in the table `trade`. The function `totalvolume` is defined in the sample Kx Systems, Inc. kdb+ file, `tradedata.q`.

## See Also

`exec` | `insert` | `kx`

**External Websites**

Kx Systems, Inc. Documentation
MATLAB client for kdb+

**Introduced in R2007a**

# get

Retrieve Kx Systems, Inc. kdb+ connection object properties

## Syntax

```
v = get(k,'PropertyName')
v = get(k)
```

## Arguments

| | |
|---|---|
| k | Kx Systems, Inc. kdb+ connection object created with kx. |
| 'PropertyName' | A character vector, string, cell array of character vectors, or string array containing property names. The property names are:<br>• 'handle'<br>• 'ipaddress'<br>• 'port' |

## Description

v = get(k,'PropertyName') returns a MATLAB structure containing the value of the specified properties for the Kx Systems, Inc. kdb+ connection object.

v = get(k) returns a MATLAB structure where each field name is the name of a property of k and the associated value of the property.

## Examples

Get the properties of the connection to the Kx Systems, Inc. kdb+ database, K:

```
v = get(k)
v =
            handle:  [1x1 c]
       ipaddress: 'localhost'
       port: '5001'
```

## See Also

close | exec | fetch | insert | kx

**Introduced in R2007a**

# insert

Write data to Kx Systems, Inc. kdb+ databases

## Syntax

```
insert(k,tablename,data)
x = insert(k,tablename,data,sync)
```

## Arguments

| | |
|---|---|
| k | The Kx Systems, Inc. kdb+ connection object created with kx. |
| tablename | The name of the Kx Systems, Inc. kdb+ Tablename. |
| data | The data that insert writes to the Kx Systems, Inc. kdb+ Tablename. |

## Description

`insert(k,tablename,data)` writes the data, `data`, to the Kx Systems, Inc. kdb+ table, `tablename`.

`x = insert(k,tablename,data,sync)` writes the data, `data`, to the Kx Systems, Inc. kdb+ table, `tablename`, synchronously. For asynchronous calls, enter `sync` as 0 (default), and for synchronous calls, enter `sync` as 1.

## Examples

For the connection to the Kx Systems, Inc. kdb+ database, `k`, write data from `ACME` to the specified table:

```
insert(k,'trade',{'`ACME',133.51,250,6.4,'2006.10.24'})
```

## See Also

`close` | `fetch` | `get` | `tables`

**External Websites**
Kx Systems, Inc. Documentation
MATLAB client for kdb+

**Introduced in R2007a**

# isconnection

Determine if connections to Kx Systems, Inc. kdb+ databases are valid

## Syntax

```
x = isconnection(k)
```

## Arguments

| | |
|---|---|
| k | Kx Systems, Inc. kdb+ connection object created with kx. |

## Description

`x = isconnection(k)` returns `x = 1` if the connection to the Kx Systems, Inc. kdb+ database is valid, and `x = 0` otherwise.

## Examples

Establish a connection to a Kx Systems, Inc. kdb+ database, k:

```
k = kx('localhost',5001);
```

Verify that k is a valid connection:

```
x = isconnection(k)
x = 1
```

## See Also

close | fetch | get | kx

**Introduced in R2007a**

# tables

Retrieve table names from Kx Systems, Inc. kdb+ databases

## Syntax

```
t = tables(k)
```

## Arguments

| | |
|---|---|
| k | The Kx Systems, Inc. kdb+ connection object created with the kx function. |

## Description

`t = tables(k)` returns the list of tables for the Kx Systems, Inc. kdb+ connection.

## Examples

Retrieve table information for the Kx Systems, Inc. kdb+ database using the connection k:

```
t = tables(k)
t =
     'intraday'
     'seclist'
     'trade'
```

## See Also

exec | fetch | insert | kx

**External Websites**

Kx Systems, Inc. Documentation
MATLAB client for kdb+

**Introduced in R2007a**

# moneynet

Create Money.Net connection

## Description

The `moneynet` function creates a `moneynet` object. The `moneynet` object represents a Money.Net connection.

After you create a `moneynet` object, you can use the object functions to retrieve current, intraday, historical, real-time, and news data. You retrieve data based on your credentials, which consist of a user name and password. For credentials, contact Money.Net.

## Creation

### Syntax

```
c = moneynet(username,password)
c = moneynet(username,password,portnumber)
```

#### Description

`c = moneynet(username,password)` creates a Money.Net connection, sets the Username property, and uses a password.

`c = moneynet(username,password,portnumber)` also sets the Port property.

#### Input Arguments

**`password` — Password**
character vector | string scalar

Password required to access Money.Net data, specified as a character vector or string scalar. To request your Money.Net password, contact Money.Net.

Data Types: `char` | `string`

### Properties

**`Username` — User name**
character vector | string scalar

User name required to access Money.Net data, specified as a character vector or string scalar. The user name is an email address. To request your Money.Net user name, contact Money.Net.

Example: `'user@company.com'`

Data Types: `char` | `string`

**`Port` — Port number**
`50010` (default) | numeric scalar

Port number of the Money.Net data server, specified as a numeric scalar.

Data Types: `double`

**`Server` — Money.Net server name**
character vector

This property is read-only.

Money.Net server name, specified as a character vector.

Example: `'NTY_JAMES_IRWIN_88 TCP'`

Data Types: `char`

## Object Functions

## Money.Net Connection

| | |
|---|---|
| close | Close Money.Net connection |
| isconnection | Determine if Money.Net connection is valid |

## Money.Net Data Retrieval

| | |
|---|---|
| getdata | Retrieve Money.Net current data |
| getsubscriptions | Retrieve Money.Net subscribed symbols and event handler functions |
| news | Search and stream Money.Net latest news stories |
| optionchain | Retrieve Money.Net option symbols |
| realtime | Retrieve Money.Net real-time data |
| stop | Unsubscribe Money.Net real-time data updates |
| timeseries | Retrieve Money.Net intraday and historical data |

## Examples

### Connect to Money.Net

Create a Money.Net connection, and then retrieve current data for a symbol.

Connect to Money.Net using a user name and password. `c` is the Money.Net connection object.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd)
```

```
c =

  moneynet with properties:

    Username: 'user@company.com'
        Port: 50010
      Server: 'NTY_JAMES_IRWIN_88 TCP'
```

Verify the Money.Net connection `c` using the `isconnection` function. This function returns `1`, indicating a successful connection.

```
v = isconnection(c)
```

```
v =

  logical

   1
```

Retrieve Money.Net current data `d` for the symbol IBM by using the Money.Net connection `c`. Specify the Money.Net data fields `f` for ask and bid price.

```
symbol = 'IBM';
f = {'Ask','Bid'};

d = getdata(c,symbol,f);
```

Close the Money.Net connection.

```
close(c)
```

### Connect to Money.Net Using Port Number

Create a Money.Net connection, and then retrieve news stories.

Connect to Money.Net using a user name, password, and port number. `c` is the Money.Net connection object.

```
username = 'user@company.com';
pwd = '999999';
portnumber = 50010;

c = moneynet(username,pwd,portnumber)
```

```
c =

  moneynet with properties:

    Username: 'user@company.com'
        Port: 50010
      Server: 'NTY_JAMES_IRWIN_88 TCP'
```

Verify the Money.Net connection `c` using the `isconnection` function. This function returns `1`, indicating a successful connection.

```
v = isconnection(c)
```

```
v =

  logical

   1
```

Retrieve news data `n` for 10 news stories by using the Money.Net connection `c`.

```
n = news(c,'Number',10);
```

Close the Money.Net connection.

```
close(c)
```

## See Also

**Topics**
“Retrieve Current and Historical Money.Net Data” on page 5-2
“Retrieve Real-Time Money.Net Data” on page 5-5
“Retrieve Money.Net News Stories” on page 5-7
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**
Money.Net API Documentation
Money.Net Help

**Introduced in R2016b**

# close

Close Money.Net connection

## Syntax

```
close(c)
```

## Description

`close(c)` closes the Money.Net connection `c`.

## Examples

### Close Money.Net Connection

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

## See Also

`getdata` | `isconnection` | `moneynet` | `news` | `realtime` | `timeseries`

### Topics

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Retrieve Real-Time Money.Net Data” on page 5-5
“Retrieve Money.Net News Stories” on page 5-7
“Money.Net Error and Warning Messages” on page 5-10

### External Websites

Money.Net API Documentation

### Introduced in R2016b

# isconnection

Determine if Money.Net connection is valid

## Syntax

```
v = isconnection(c)
```

## Description

`v = isconnection(c)` returns logical `1 (true)` if `c` is a valid Money.Net connection. Otherwise, `isconnection` returns logical `0 (false)`.

## Examples

### Validate Money.Net Connection

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Validate the Money.Net connection `c`.

```
v = isconnection(c)
```

```
v =

  logical

   1
```

`isconnection` returns `1`, indicating a successful connection.

Close the Money.Net connection.

```
close(c)
```

Validate that the Money.Net connection `c` is closed.

```
v = isconnection(c)
```

```
v =

  logical

   0
```

`isconnection` returns `0`, indicating a closed connection.

## Input Arguments

### c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

## Output Arguments

### v — Valid Money.Net connection

`true` | `false`

Valid Money.Net connection, returned as a logical `1 (true)` that specifies a successful connection, or logical `0 (false)` that specifies a closed or invalid connection.

## See Also

`close` | `moneynet`

**Topics**

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Retrieve Real-Time Money.Net Data” on page 5-5
“Retrieve Money.Net News Stories” on page 5-7
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# getdata

Retrieve Money.Net current data

## Syntax

```
d = getdata(c,symbols,f)
```

## Description

`d = getdata(c,symbols,f)` returns Money.Net data `d` using the Money.Net connection `c` for the symbols and the Money.Net fields `f`.

## Examples

### Retrieve Current Data for One Symbol

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve Money.Net current data `d` for the symbol IBM using the Money.Net connection `c`. Specify the Money.Net data fields `f` for ask and bid price.

```
symbol = 'IBM';
f = {'Ask','Bid'};

d = getdata(c,symbol,f);
```

Display Money.Net current data.

```
d
```

```
d =

    Symbol      Ask        Bid
    ______    _______    _______

    'IBM'     145.00     143.85
```

`d` is a table that contains the columns for symbol, ask price, and bid price. The row contains the Money.Net data values for each column.

Close the Money.Net connection.

```
close(c)
```

### Retrieve Current Data for Multiple Symbols

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve Money.Net current data `d` for the `symbols` list that contains IBM, Google, and Yahoo! using the Money.Net connection `c`. Specify the Money.Net data fields `f` for ask and bid price.

```
symbols = {'IBM','GOOG','YHOO'};
f = {'Ask','Bid'};

d = getdata(c,symbols,f);
```

Display Money.Net current data.

```
d
```

```
d =

    Symbol      Ask        Bid
    ______    _______    _______

    'IBM'     145.00     143.85
    'GOOG'    700.50     700.05
    'YHOO'     37.50      37.41
```

`d` is a table that contains the columns for symbol, ask price, and bid price. The rows contains the Money.Net data values for each symbol in the symbol list.

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### `c` — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### `symbols` — Money.Net symbol list

character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char` | `cell` | `string`

### f — Money.Net data field list

character vector | cell array of character vectors | string scalar | string array

Money.Net data field list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one field, use a character vector or string scalar. To specify multiple fields, use a cell array of character vectors or a string array.

Specify the field by using the single character or the field definition. For example, to specify the highest price for the equity during the current trading day, use a single character `'H'` or the corresponding field definition `'High'`. When using the field definition, the software ignores the case of the definition. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation.

Example: `'High'`

Example: `{'High','Low'}`

Data Types: `char` | `cell` | `string`

## Output Arguments

### d — Money.Net data

table

Money.Net data, returned as a table. Each row corresponds to the `symbols` list. Each column corresponds to the field list `f`.

## See Also

`close` | `moneynet` | `news` | `realtime` | `timeseries`

**Topics**

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# timeseries

Retrieve Money.Net intraday and historical data

## Syntax

```
d = timeseries(c,s,date,interval)
d = timeseries(c,s,date,interval,f)
```

## Description

`d = timeseries(c,s,date,interval)` returns Money.Net intraday and historical data using the Money.Net connection `c` for all available fields. Specify the Money.Net symbol `s` and the current or historical date. To specify the amount of data to return, use the bar interval.

`d = timeseries(c,s,date,interval,f)` returns Money.Net intraday and historical data for the specified Money.Net fields `f`.

## Examples

### Retrieve Intraday Data in Seconds

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve intraday data for the last 5 minutes in 30-second bars for the symbol IBM using the Money.Net connection `c`. Specify the date as a `datetime` array containing a date range with start and end dates. The start date starts 5 minutes after the current moment. The end date is the current moment. To specify the current moment, use `datetime('now')`. To specify 5 minutes earlier, subtract `minutes(5)` from the current moment. To retrieve data in 30-second bars, specify the interval as `'30S'`.

```
s = 'IBM';
date = [datetime('now')-minutes(5) datetime('now')];
interval = '30S';

d = timeseries(c,s,date,interval);
```

Display the first three rows of intraday data `d` for all valid Money.Net fields.

```
d(1:3,:)
```

```
ans =

          Date             High       Low        Open       Close      Volume
    ___________________    _______    _______    _______    _______    ________
```

```
    05/09/16 13:30:30    147.52    147.48    147.48    147.51    2763.00
    05/09/16 13:31:00    147.53    147.50    147.50    147.52    7241.00
    05/09/16 13:31:30    147.54    147.51    147.51    147.53    5608.00
```

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price
- Open price
- Close price
- Trading volume

Close the Money.Net connection.

```
close(c)
```

### Retrieve Intraday Data in Minutes

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve intraday data for yesterday in 30-minute bars for the symbol IBM using the Money.Net connection `c`. Specify the date as yesterday using `datetime`. To retrieve data in 30-minute bars, specify the interval as `'30M'`.

```
s = 'IBM';
date = datetime('yesterday');
interval = '30M';

d = timeseries(c,s,date,interval);
```

Display the first three rows of intraday data `d` for all valid Money.Net fields.

```
d(1:3,:)
```

```
ans =

          Date             High      Low       Open      Close     Volume
    ___________________    ______    ______    ______    ______    _______

    05/06/16 08:00:00    145.22    145.07    145.07    145.22    2455.00
    05/06/16 08:30:00    144.66    144.66    144.66    144.66     300.00
    05/06/16 09:00:00    145.00    144.90    144.90    145.00    4758.00
```

`d` is a table that contains these columns:

- Date timestamp
- High price

- Low price
- Open price
- Close price
- Trading volume

Close the Money.Net connection.

```
close(c)
```

### Retrieve Daily Historical Data for Specified Fields

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve historical data in daily bars for the symbol IBM using the Money.Net connection `c`. Specify the date range from June 1, 2015 through June 5, 2015 using `datetime`. To retrieve daily data, specify the interval as `'1D'`. Retrieve only the high and low price fields `f` from Money.Net.

```
s = 'IBM';
date = [datetime('1-Jun-2015') datetime('5-Jun-2015')];
interval = '1D';
f = {'High','Low'};

d = timeseries(c,s,date,interval,f);
```

Display the first three rows of daily data `d`.

```
d(1:3,:)
```

```
ans =

          Date             High      Low
    ___________________    ______    ______

    06/01/15 00:00:00    171.04    169.03
    06/02/15 00:00:00    170.45    168.43
    06/03/15 00:00:00    171.56    169.63
```

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price

Close the Money.Net connection.

```
close(c)
```

### Retrieve Weekly Historical Data for Specified Fields

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve historical data in weekly bars for the symbol IBM using the Money.Net connection `c`. Specify the date range from June 1, 2015 through June 30, 2015 using `datetime`. To retrieve weekly data, specify the interval as `'7D'`. Retrieve only the high and low price fields `f` from Money.Net.

```
s = 'IBM';
date = [datetime('1-Jun-2015') datetime('30-Jun-2015')];
interval = '7D';
f = {'High','Low'};

d = timeseries(c,s,date,interval,f);
```

Display the first three rows of weekly data `d`.

```
d(1:3,:)
```

```
ans =

          Date             High      Low
    ___________________    ______    ______

    06/01/15 00:00:00    171.56    167.20
    06/08/15 00:00:00    170.44    163.37
    06/15/15 00:00:00    168.72    164.25
```

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### s — Money.Net symbol

character vector | cell array of character vector | string scalar

Money.Net symbol, specified as a character vector, cell array of a character vector, or string scalar to denote one symbol.

Example: `'IBM'`

Data Types: `char` | `cell` | `string`

**`date` — Date**
`datetime` array | character vector | cell array of character vectors | double | string scalar | string array

Date, specified as a `datetime` array, character vector, cell array of character vectors, double, string scalar, or string array. If `date` contains one date, this date is the start date. The software determines the end date to be the last second of the same day. If `date` contains two dates, the first date is the start date and the second date is the end date.

Example: `datetime('yesterday')`

Data Types: `datetime` | `char` | `cell` | `double` | `string`

**`interval` — Interval**
character vector | string scalar

Interval between bars, specified as a character vector or string scalar. Specify the interval as a number followed by one of these letters: `S`, `M`, and `D`. These letters indicate seconds, minutes, and days, respectively. For example, `30S` is 30-second bars and `1D` is daily end-of-day data.

Data Types: `char` | `string`

**`f` — Money.Net data field list**
character vector | cell array of character vectors | string scalar | string array

Money.Net data field list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one field, use a character vector or string scalar. To specify multiple fields, use a cell array of character vectors or a string array.

Specify the field by using the single character or the field definition. For example, to specify the highest price for the equity during the current trading day, use a single character `'H'` or the corresponding field definition `'High'`. When using the field definition, the software ignores the case of the definition. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation.

Example: `'High'`

Example: `{'High','Low'}`

Data Types: `char` | `cell` | `string`

## Output Arguments

**`d` — Money.Net data**
table

Money.Net data, returned as a table. Each row in the table represents data at different times. The first column `Date` is the timestamp. The remaining columns contain one column of data for each Money.Net field `f`.

To return data for all available historical fields, use this syntax:

```
d = timeseries(c,s,date,interval);
```

Money.Net returns data only for business days with trading activity.

## See Also

`close` | `getdata` | `moneynet` | `news` | `realtime`

**Topics**

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# realtime

Retrieve Money.Net real-time data

## Syntax

```
realtime(c,symbols)
realtime(c,symbols,eventhandler)
```

## Description

`realtime(c,symbols)` subscribes to real-time data updates using the Money.Net connection `c` for the specified symbols. The default event handler function `mnRealTimeEventHandler` processes and retrieves real-time data updates for each specified symbol.

`realtime(c,symbols,eventhandler)` processes real-time data updates using a custom event handler function `eventhandler`.

## Examples

### Retrieve Money.Net Real-Time Data for One Symbol

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve Money.Net real-time data updates for the IBM symbol.

```
symbol = 'IBM';

realtime(c,symbol)
```

The default event handler `mnRealTimeEventHandler` processes all real-time data updates. To access the code for the default event handler, enter `edit mnRealTimeEventHandler.m`.

`mnRealTimeEventHandler` creates the workspace variable `IBMRealTime`. The `mnRealTimeEventHandler` function populates the table `IBMRealTime` with real-time data updates. To see the real-time data, open `IBMRealTime` in the Variables editor.

Stop the symbol subscription.

```
stop(c)
```

`mnRealTimeEventHandler` stops processing all real-time data updates. The last real-time data update remains in `IBMRealTime`.

Close the Money.Net connection.

```
close(c)
```

### Retrieve Money.Net Real-Time Data for Multiple Symbols

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve Money.Net real-time data updates for the symbols IBM and Yahoo!.

```
symbols = {'IBM','YHOO'};

realtime(c,symbols)
```

The default event handler `mnRealTimeEventHandler` processes all real-time data updates. To access the code for the default event handler, enter `edit mnRealTimeEventHandler.m`.

The `mnRealTimeEventHandler` function creates the workspace variables `IBMRealTime` and `YHOORealTime`. The `mnRealTimeEventHandler` function populates the tables `IBMRealTime` and `YHOORealTime` with real-time data updates. To see the real-time data, open either variable in the Variables editor.

Stop all symbol subscriptions.

```
stop(c)
```

`mnRealTimeEventHandler` stops processing all real-time data updates. The last real-time data update remains in each workspace variable.

Close the Money.Net connection.

```
close(c)
```

### Retrieve Money.Net Real-Time Data Using a Custom Event Handler

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Define a custom event handler function `myfcn`. The `myfcn` function displays real-time Money.Net data to the Command Window. You can write a custom function that processes real-time data updates differently. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

```
myfcn = @(x)disp(x);
```

Retrieve Money.Net real-time data updates for the IBM symbol using `myfcn`.

```
symbol = 'IBM';

realtime(c,symbol,myfcn)
```

| Symbol | Description | Yesterday | YesterdayDateTime | Bid | Ask | ExchangeOfTheCurrentBidPrice | Ex |
|---|---|---|---|---|---|---|---|
| 'IBM' | 'INTERNATIONAL BUSINESS MACHS' | 148.31 | 05/24/16 00:00:00 | 151.65 | 151.67 | '' | '' |

`myfcn` displays real-time data updates for IBM in the Command Window.

Stop the symbol subscription.

```
stop(c)
```

`myfcn` stops displaying real-time data updates in the Command Window.

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### `c` — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### `symbols` — Money.Net symbol list

character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char` | `cell` | `string`

### `eventhandler` — Event handler

`'mnRealTimeEventHandler'` (default) | character vector | string scalar | function handle

Event handler, specified as a character vector, string scalar, or a function handle that specifies the name of the event handler function. Write a custom event handler function to process any type of real-time Money.Net events. This function must have at least one input argument that is a table. The table format must be similar to the format of the output argument in `getdata`. The event handler function returns all available fields when it executes for the first time. The event handler function executes every time Money.Net provides a real-time update. For details about custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-26.

For example, to display real-time data updates in the Command Window, enter this code to define a custom event handler function:

```
symbol = 'IBM';
myfcn = @(x)disp(x);

realtime(c,symbol,myfcn)
```

If you do not specify a custom event handler function, the default event handler `mnRealTimeEventHandler` runs. To access the code for the default event handler, enter `edit mnRealTimeEventHandler.m`.

The `mnRealTimeEventHandler` function creates a workspace variable. The workspace variable name is a concatenation of the symbol name and the word `RealTime`. For example, `mnRealTimeEventHandler` populates real-time data for the symbol IBM into `IBMRealTime`. This workspace variable is a table with columns for each field. The values in the table change when Money.Net provides a real-time data update. Empty fields from Money.Net populate as `NaN`, `NaT`, and so on, depending on the data type.

First, `mnRealTimeEventHandler` runs using a table of current data. Then, `mnRealTimeEventHandler` runs each time an update occurs.

Data Types: `char` | `function_handle` | `string`

## See Also

`close` | `getdata` | `getsubscriptions` | `moneynet` | `news` | `stop`

**Topics**

“Retrieve Real-Time Money.Net Data” on page 5-5
“Writing and Running Custom Event Handler Functions” on page 1-26
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# stop

Unsubscribe Money.Net real-time data updates

## Syntax

```
stop(c)
stop(c,symbols)
```

## Description

`stop(c)` unsubscribes real-time data updates associated with the Money.Net connection `c`.

`stop(c,symbols)` unsubscribes real-time data updates for the specified symbols.

## Examples

### Unsubscribe All Real-Time Data Updates

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

```
symbols = {'IBM','YHOO'};

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Unsubscribe from all symbol subscriptions.

```
stop(c)
```

The default event handler function stops processing all real-time data updates. For details about the event handler function, see `realtime`.

Close the Money.Net connection.

```
close(c)
```

### Unsubscribe Real-Time Data Updates for One Symbol

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

```
symbols = {'IBM','YHOO'};

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Unsubscribe from real-time data updates for IBM only.

```
symbol = 'IBM';

stop(c,symbol)
```

The default event handler function stops processing real-time data updates for IBM. The real-time data updates continue for Yahoo! only. For details about the event handler function, see `realtime`.

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### `c` — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### `symbols` — Money.Net symbol list

character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char` | `cell` | `string`

## See Also

`close` | `getsubscriptions` | `moneynet` | `realtime`

**Topics**

“Retrieve Real-Time Money.Net Data” on page 5-5
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# getsubscriptions

Retrieve Money.Net subscribed symbols and event handler functions

## Syntax

```
subs = getsubscriptions(c)
```

## Description

`subs = getsubscriptions(c)` returns the subscription list `subs` that contains open subscriptions for the Money.Net connection `c`.

## Examples

### Retrieve Subscribed Symbols and Event Handlers

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

```
symbols = {'IBM','YHOO'};

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Retrieve the subscribed symbols and the corresponding event handler function for each symbol using the Money.Net connection `c`.

```
subs = getsubscriptions(c)
```

```
subs =

    Symbols         EventHandlers
    _______    ____________________________

    'IBM'      @mnRealTimeEventHandler
    'YHOO'     @mnRealTimeEventHandler
```

`subs` returns a table with a row for each symbol and the corresponding event handler function.

Unsubscribe from all symbols using the Money.Net connection `c`.

```
stop(c)
```

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

## Output Arguments

### subs — Subscription list

table

Subscription list, returned as a table. The list contains all currently subscribed symbols and the corresponding event handler function that is processing real-time updates for each symbol. Each row in the table represents one unique subscription.

If there are no subscribed symbols, `subs` is an empty table.

## See Also

`close` | `moneynet` | `realtime` | `stop`

**Topics**

“Retrieve Real-Time Money.Net Data” on page 5-5
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# optionchain

Retrieve Money.Net option symbols

## Syntax

```
o = optionchain(c,s)
```

## Description

`o = optionchain(c,s)` returns the option symbols using the Money.Net connection `c` and symbol `s`.

## Examples

### Retrieve Option Symbols for Specified Symbol

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve option symbols `o` for the symbol IBM.

```
s = 'IBM';

o = optionchain(c,s);
```

`o` is a cell array of character vectors. Each character vector is an option symbol.

Display the first three option symbols.

```
o(1:3)
```

```
ans =

  3×1 cell array

    'O:IBM\16Q13\130 .0'
    'O:IBM\16E27\148 .0'
    'O:IBM\16Q20\138 .0'
```

Retrieve current data for the first option symbol `o(1)` and display it. Specify fields `f` for describing the option symbol:

- Option symbol description
- Option symbol strike
- Option symbol expiration date

```
symbol = o(1);
f = {'Description','Strike','Expiration'};

d = getdata(c,symbol,f)
```

```
d =

          Symbol                 Description             Strike    Expiration
    ____________________    ____________________________    ______    __________

    'O:IBM\16F24\131 .0'    'IBM Call 06/24/2016 131.0'    131       06/24/16
```

`d` is a table with one row of data. The data contains the option symbol name in the first column and a column for each specified field `f`.

To retrieve intraday data, use `timeseries`.

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### s — Money.Net symbol

character vector | cell array of character vector | string scalar

Money.Net symbol, specified as a character vector, cell array of a character vector, or string scalar to denote one symbol.

Example: `'IBM'`

Data Types: `char` | `cell` | `string`

## Output Arguments

### o — Option symbols

cell array of character vectors

Option symbols, returned as a cell array of character vectors. Each character vector specifies one option symbol. The total number of option symbols depends on the symbol `s`.

## See Also

`close` | `getdata` | `moneynet` | `timeseries`

### Topics

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Money.Net Error and Warning Messages” on page 5-10

### External Websites

Money.Net API Documentation

**Introduced in R2016b**

# news

Search and stream Money.Net latest news stories

## Syntax

```
n = news(c)
n = news(c,Name,Value)
news(c,Name,Value)
```

## Description

`n = news(c)` returns Money.Net news stories `n` using the Money.Net connection `c`.

`n = news(c,Name,Value)` returns news stories with additional options specified by one or more `Name,Value` pair arguments.

`news(c,Name,Value)` streams news stories in real time using the streaming options.

## Examples

### Retrieve News Stories

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve news data `n` for 50 news stories using the Money.Net connection `c`.

```
n = news(c);
```

`n` returns as a table with 50 rows.

Display the news story title, identifier, and published time for the first news story in the table `n`.

```
n(1,1:3)
```

```
ans =

                          ArticleTitle                           ArticleID       PublishedTime
    _____________________________________________________________    _________    ________________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09    05/13/16 10:00:02
```

Close the Money.Net connection.

```
close(c)
```

### Retrieve a Specific Number of Stories

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve news data `n` for 10 news stories using the Money.Net connection `c`.

```
n = news(c,'Number',10);
```

`n` returns as a table with 10 rows.

Display the news story title, identifier, and published time for the first news story in the table `n`.

```
n(1,1:3)
```

```
ans =
                            ArticleTitle                                ArticleID       PublishedTime
    ____________________________________________________________________    _________    ________________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09    05/13/16 10:00:02
```

Close the Money.Net connection.

```
close(c)
```

### Filter News Story Retrieval by Specific Criteria

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Retrieve news stories in the general finance category. Specify that the news stories mention the term `'Dropbox'` and contain the symbol for IBM.

```
category = 'General Finance';
term = 'Dropbox';
symbol = 'IBM';

n = news(c,'Category',category,'SearchTerm',term,'Symbol',symbol);
```

`n` is a table with one news story.

Display the news story title, identifier, and published time for the news story.

```
n(1,1:3)
```

```
ans =
                        ArticleTitle                          ArticleID       PublishedTime
    ______________________________________________________    _________    ________________

    'Hewlett Packard Enterprise (HPE) Teams Up with Dropbox'    4.0002e+09    06/08/16 11:11:05
```

Close the Money.Net connection.

```
close(c)
```

### Stream News Data

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Turn on the subscription to the Money.Net real-time news data stream using the default event handler function `mnNewsStreamEventHandler`. The function `mnNewsStreamEventHandler` processes news data events by populating the workspace variable `mnNewsStreamLatest` with the latest news stories. News stories populate in the `mnNewsStreamLatest` variable until it contains 10 rows. Then, the latest news stories overwrite the older ones in `mnNewsStreamLatest`. To access the code for this function, enter `edit mnNewsStreamEventHandler.m`.

```
news(c,'Subscription','on')
```

The workspace variable `mnNewsStreamLatest` appears in the MATLAB Workspace.

Display the news story title, identifier, and published time for the first news story.

```
mnNewsStreamLatest(1,1:3)
```

```
ans =

                          ArticleTitle                                 ArticleID       PublishedTime
    ___________________________________________________________________    _________    _________________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09    05/13/16 10:00:02
```

To see the latest 10 news stories, explore `mnNewsStreamLatest` in the Variables editor.

Turn off the real-time news data stream.

```
news(c,'Subscription','off')
```

Real-time updates stop in the workspace variable `mnNewsStreamLatest`.

Close the Money.Net connection.

```
close(c)
```

### Stream News Data Using Custom Event Handler

Create Money.Net connection `c` using a user name and password.

```
username = 'user@company.com';
pwd = '999999';

c = moneynet(username,pwd);
```

Turn on the subscription to the Money.Net real-time news data stream using the custom event handler function `myfnc`. Here, define `myfnc` to display Money.Net news data to the Command Window. You can write a custom event handler function to process streaming news stories differently. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

```
myfnc = @(x)disp(x);

news(c,'Subscription','on','EventHandler',myfnc)
```

```
                          ArticleTitle                                ArticleID      PublishedTime      PublisherCode    Publisher
    ______________________________________________________________    _________    _________________    _____________    _________

    '@ETFcom: The Most Important ETF Of 2016 https://t.co/a5qCYK2o7c …'    3.9089e+09    05/17/16 14:39:10    'TWIT'           'Twitter'
```

Money.Net news stories stream to the Command Window.

Turn off the real-time news data stream.

```
news(c,'Subscription','off')
```

Real-time updates stop in the Command Window.

Close the Money.Net connection.

```
close(c)
```

## Input Arguments

### `c` — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `n = news(c,'Number',10);`

---

**Note** The name-value pair arguments in the searching and streaming groups are independent. If you combine these name-value pair arguments, you receive this error: `Invalid combination of Name-Value pairs. Type HELP MONEYNET/NEWS to see the valid syntax`.

---

#### Searching News Stories Options

#### `Number` — Number of news stories

`50` (default) | numeric scalar

Number of news stories, specified as the comma-separated pair consisting of `'Number'` and a numeric scalar. The maximum number of news stories that the Money.Net API can return is 500.

The number of news stories returned can be fewer than the specified number because Money.Net provides only available news stories. When you specify this option by itself, `news` does not filter the story content.

Example: `n = news(c,'Number',10);`

Data Types: `double`

### `SearchTerm` — Search term

character vector | string scalar

Search term, specified as the comma-separated pair consisting of `'SearchTerm'` and a character vector or a string scalar. `news` returns available news stories that contain the search term in the title or body of the news story.

Example: `n = news(c,'SearchTerm','Windows 10');`

Data Types: `char` | `string`

### `Symbol` — Symbol

character vector | cell array of character vectors | string scalar | string array

Symbol, specified as the comma-separated pair consisting of `'Symbol'` and a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array. `news` returns news stories related to the specified symbols.

Example: `n = news(c,'Symbol',{'IBM,'YHOO'});`

Data Types: `char` | `cell` | `string`

### `Category` — News category

character vector | string scalar

News category, specified as the comma-separated pair consisting of `'Category'` and a character vector or a string scalar. `news` returns stories only in the news category specified.

Example: `n = news(c,'Category','General Finance');`

Data Types: `char` | `string`

#### Streaming News Stories Options

### `Subscription` — Money.Net real-time news subscription

`'on'` | `'off'`

Money.Net real-time news subscription, specified as the comma-separated pair consisting of `'Subscription'` and the values `'on'` or `'off'`. To turn on the Money.Net real-time news subscription, specify the value `'on'`. To turn off the subscription, specify the value `'off'`.

By default, the sample event handler function `mnNewsStreamEventHandler` processes the retrieval of news stories during real-time news subscription. To access the code for this function, enter `edit mnNewsStreamEventHandler.m`. The `mnNewsStreamEventHandler` function creates the workspace variable `mnNewsStreamLatest`. Then, `mnNewsStreamEventHandler` populates the table `mnNewsStreamLatest` with the latest 10 news stories from Money.Net. Then, the `mnNewsStreamEventHandler` function updates the list to display the latest news stories.

To specify a custom event handler function, use the name-value pair argument `'EventHandler'`.

Example: `news(c,'Subscription','on')`

Example: `news(c,'Subscription','on','EventHandler',myFcn)`

**`EventHandler` — Custom event handler function**
character vector | string scalar | function handle

Custom event handler function, specified as the comma-separated pair consisting of `'EventHandler'` and a character vector, string scalar, or function handle. To process the latest news stories, you can write your own custom event handler function. This function must have an input argument specified as a table. Each new news story from Money.Net is a single row in a table. For details about working with custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-26.

Specify this name-value pair argument only with the name-value pair argument `'Subscription'` and value `'on'`.

Example: `news(c,'Subscription','on','EventHandler',myFcn)`

Data Types: `char` | `function_handle` | `string`

## Output Arguments

**`n` — News stories**
table

News stories, returned as a table with these variables. Each row in the table represents one news story. For details about these variables, see Money.Net API Documentation.

| News Story Variable | Data Type | Variable Description |
|---|---|---|
| `ArticleTitle` | cell array of character vectors | News story title |
| `ArticleID` | double | Internal Money.Net identifier of the news story |
| `PublishedTime` | `datetime` array | Date and time the news story was published |
| `PublisherCode` | cell array of character vectors | Publisher four-digit code |
| `PublisherName` | cell array of character vectors | Publisher name |
| `ArticleBodyDescription` | cell array of character vectors | Body excerpt or short description of the news story |
| `Category` | cell array of character vectors | Internal Money.Net category of the news story |
| `URLLink` | cell array of character vectors | Website that contains the full news story |
| `Source` | cell array of character vectors | News stream source code |
| `Priority` | cell array of character vectors | Priority of news story with these values:<br>• `0` is Normal<br>• `1` is Breaking<br>• `2` is Major Hot |

| News Story Variable | Data Type | Variable Description |
|---|---|---|
| `Symbols` | cell array | Symbols, or tickers, associated with the news story with these values:<br>• An empty cell array for no symbols<br>• A cell array that contains the symbol as a character vector for one symbol<br>• A nested cell array that contains symbols as character vectors for multiple symbols |

## See Also

`close` | `getdata` | `moneynet` | `realtime` | `timeseries`

**Topics**

“Retrieve Money.Net News Stories” on page 5-7
“Writing and Running Custom Event Handler Functions” on page 1-26
“Money.Net Error and Warning Messages” on page 5-10

**External Websites**

Money.Net API Documentation

**Introduced in R2016b**

# ravenpack

RavenPack News Analytics connection

## Description

The `ravenpack` function creates a `ravenpack` object. The `ravenpack` object represents a RavenPack News Analytics connection.

After you create a `ravenpack` object, you can use the object functions to retrieve intraday, historical, and real-time news event data.

To use the RavenPack News Analytics software, you must first download it from the RavenPack Developer Zone.

## Creation

### Syntax

```
c = ravenpack(user,password)
c = ravenpack(user,password,application)
```

#### Description

`c = ravenpack(user,password)` connects to RavenPack News Analytics Data Gateway, sets the User property, and uses a password.

`c = ravenpack(user,password,application)` also sets the Application property.

#### Input Arguments

**`password` — Password**
character vector | string scalar

Password, specified as a character vector or string scalar. For your password, contact RavenPack.

Data Types: `char` | `string`

### Properties

**`Application` — RavenPack application service**
`'rpna-api'` (default) | character vector | string scalar

This property is read-only.

RavenPack application service, specified as a character vector or string scalar.

Data Types: `char` | `string`

### `Handle` — RavenPack Data Gateway Client

RavenPack Data Gateway Client object

This property is read-only.

RavenPack Data Gateway Client, specified as a RavenPack Data Gateway Client object.

Example: `[1x1 com.ravenpack.data.DataGatewayClient]`

### `User` — User name

character vector | string scalar

This property is read-only.

User name, specified as a character vector or string scalar. For your user name, contact RavenPack.

Example: `'username'`

Data Types: `char` | `string`

### `DataReturnFormat` — Data return format

`'table'` (default) | `'timetable'`

Data return format, specified as one of these values, which determine the data type of the returned data.

| Value | Data Type of Returned Data |
|---|---|
| `'table'` | `table` |
| `'timetable'` | `timetable` |

You can specify these values using a character vector or string (for example, `"table"`).

To retrieve data as a timetable, you must set this property value manually at the command line or in a script using dot notation, for example:

```
c.DataReturnFormat = 'timetable';
```

The default date format is a `datetime` array.

After setting the `DataReturnFormat` property, use the `timeseries` function to retrieve intraday and historical data.

## Object Functions

| | |
|---|---|
| entitlements | RavenPack News Analytics Data Gateway entitlements |
| realtime | RavenPack News Analytics real-time data |
| timeseries | RavenPack News Analytics intraday and historical data |
| close | Close RavenPack News Analytics connection |

## Examples

### Connect to RavenPack

Create a RavenPack News Analytics connection. Then, retrieve historical data for a company.

Start the RavenPack Data Gateway process from the RavenPack folder in the Windows Start menu.

Add the full path of the Data Gateway Client JAR file to the dynamic Java class path. For details about static and dynamic class paths, see “Java Class Path”.

```
javaaddpath 'C:\Program Files\RavenPack\api\DataGatewayClient.jar'
```

---

**Note** If you are running Linux®, start the RavenPack Data Gateway using the following command. This command assumes that you installed RavenPack News Analytics in the default `/opt/ravenpack` folder.

```
/opt/ravenpack/bin/DataGateway.sh
```

Then, add the full path of the Data Gateway Client JAR file using this command.

```
javaaddpath '/opt/ravenpack/api/DataGatewayClient.jar'
```

---

Create a RavenPack News Analytics connection using the user name `username` and password `pwd`. The RavenPack News Analytics connection object `c` appears in the MATLAB workspace.

```
c = ravenpack('username','pwd')
```

```
c =

  ravenpack with properties:

            Application: 'rpna-api'
                 Handle: [1x1 com.ravenpack.data.DataGatewayClient]
                   User: 'username'
       DataReturnFormat: 'table'
```

Retrieve RavenPack News Analytics data for the last day. Here, the symbol is the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is 1 day ago. The end date is the current date and time. `d` is a table that contains the RavenPack News Analytics data.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-1;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

To retrieve more than 3 days of historical news data, use the RavenPack News Analytics Data Feed Tool.

Display the first four columns of the first record of historical data. Each row in the table is one record of news data. Here, the first four columns specify a news event on May 9, 2018 about a company.

```
d(1,1:4)
```

```
ans =

  1×4 table
```

```
      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    ___________________    ____________    ___________    ____________

    09-May-2018 15:02:00       2491BC          COMP        Twitter Inc.
```

Close the RavenPack News Analytics connection.

```
close(c)
```

### Connect to RavenPack Using Application Service

Create a RavenPack News Analytics connection using a RavenPack application service. Then, retrieve historical data for a company.

Start the RavenPack Data Gateway process from the RavenPack folder in the Windows Start menu.

Add the full path of the Data Gateway Client JAR file to the dynamic Java class path. For details about static and dynamic class paths, see “Java Class Path”.

```
javaaddpath 'C:\Program Files\RavenPack\api\DataGatewayClient.jar'
```

---

**Note** If you are running Linux, start the RavenPack Data Gateway using the following command. This command assumes that you installed RavenPack News Analytics in the default `/opt/ravenpack` folder.

```
/opt/ravenpack/bin/DataGateway.sh
```

Then, add the full path of the Data Gateway Client JAR file using this command.

```
javaaddpath '/opt/ravenpack/api/DataGatewayClient.jar'
```

---

Create a RavenPack News Analytics connection using the user name `username`, password `pwd`, and application service `app-service-name`. The RavenPack News Analytics connection object `c` appears in the MATLAB workspace.

```
user = 'username';
password = 'pwd';
application = 'app-service-name';
c = ravenpack(user,password,application)
```

```
c =

  ravenpack with properties:

            Application: 'app-service-name'
                 Handle: [1x1 com.ravenpack.data.DataGatewayClient]
                   User: 'username'
       DataReturnFormat: 'table'
```

Retrieve RavenPack News Analytics data for the last day. Here, the symbol is the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is 1 day ago. The end date is the current date and time. `d` is a table that contains the RavenPack News Analytics data.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-1;
```

```
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

To retrieve more than 3 days of historical news data, use the RavenPack News Analytics Data Feed Tool.

Display the first four columns of the first record of historical data. Each row in the table is one record of news data. Here, the first four columns specify a news event on May 9, 2018 about a company.

```
d(1,1:4)
```

```
ans =

  1×4 table

      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE         ENTITY_NAME
    ____________________    ____________    ___________    ____________________

    09-May-2018 15:29:20       375C05          COMP        Ecolomondo Corp. Inc.
```

Close the RavenPack News Analytics connection.

```
close(c)
```

## See Also

### Topics

“Workflow for RavenPack News Analytics” on page 3-24
“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21

### External Websites

RavenPack Developer Zone Overview

### Introduced in R2015b

# entitlements

RavenPack News Analytics Data Gateway entitlements

## Syntax

```
e = entitlements(c)
```

## Description

`e = entitlements(c)` retrieves the RavenPack News Analytics entitlements using the RavenPack News Analytics connection `c`.

## Examples

### Retrieve RavenPack News Analytics Data Gateway Entitlements

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics Data Gateway entitlements `e`.

```
e = entitlements(c);
```

Display the first two rows and columns of the entitlement data.

```
e(1:2,1:2)
```

```
ans =

          DISPLAYNAME                      NAME
    ___________________________    ____________________________________

    'RPNA-4.0-EQUITIES'         '(entity-scores :rpna-4.0-eqt)'
    'RPNA-4.0-GLOBAL-MACRO'     '(entity-scores :rpna-4.0-mgp)'
```

`e` is a table that contains the entitlement data. These columns specify the name of the RavenPack symbols. You can retrieve RavenPack News Analytics data for the RavenPack symbols specified in each row of the table.

For details about the column names in the table, see the RavenPack Developer Zone Overview.

List all the columns in the entitlement data.

```
e.Properties.VariableNames
```

```
ans =

  Columns 1 through 6

    'DISPLAYNAME'    'NAME'    'VERSION'    'SUFFIX'    'FIELDS'    'PRODSEQ'

  Column 7
```

```
'PRODUCT'
```

Close the RavenPack News Analytics connection.

```
close(c)
```

## Input Arguments

### c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

## Output Arguments

### e — RavenPack News Analytics Data Gateway entitlements

table

RavenPack News Analytics Data Gateway entitlements, returned as a table.

## Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

## See Also

`close` | `ravenpack` | `timeseries`

### Topics

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21
“Workflow for RavenPack News Analytics” on page 3-24

### External Websites

RavenPack Developer Zone Overview

### Introduced in R2015b

# realtime

RavenPack News Analytics real-time data

## Syntax

```
[status,lhandle] = realtime(c,symbol,listener)
```

## Description

`[status,lhandle] = realtime(c,symbol,listener)` retrieves RavenPack News Analytics real-time data using the RavenPack News Analytics connection `c`.

## Examples

### Retrieve RavenPack News Analytics Real-Time Data

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics real-time data using RavenPack News Analytics connection `c`. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. Use the existing event listener `rpExampleListener`. Specify these fields as inputs to `rpExampleListener`:

- `ENTITY_NAME`
- `CATEGORY`
- `SUB_TYPE`

You can modify `rpExampleListener` or create your own event listener to add other functionality.

The MATLAB variable `evt` is an instance of a news event. Do not instantiate this variable. Assign this variable any name.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
fields = {'ENTITY_NAME','CATEGORY','SUB_TYPE'};

[status,lhandle] = realtime(c,symbol,...
                            @(~,evt)rpExampleListener(evt,fields))
```

```
status =

     1

lhandle =

    handle.listener
```

`status` returns a `1` to signify a successful RavenPack News Analytics connection.

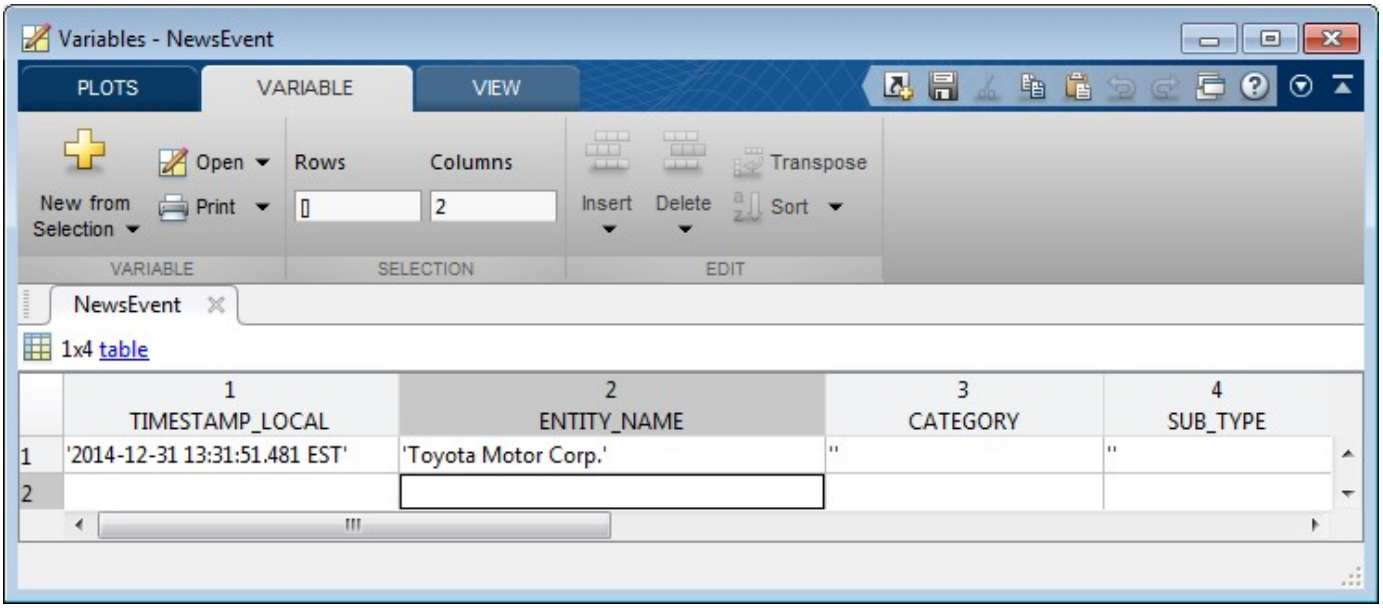
`lhandle` returns the contents of the handle to the RavenPack listener object.

The existing event listener `rpExampleListener` populates the MATLAB variable `NewsEvent` with the real-time data.

Display the real-time data.

```
openvar('NewsEvent')
```

MATLAB displays the Variables editor to show the news event data.




To stop real-time data updates, use the MATLAB `delete` function to delete the handle to the RavenPack listener object.

```
delete(lhandle)
```

Close the RavenPack News Analytics connection.

```
close(c)
```

## Input Arguments

### c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

### symbol — RavenPack entitled symbol list

character vector | string scalar | cell array of character vectors | string array

RavenPack entitled symbol list, specified as a character vector, string scalar, cell array of character vectors, or string array.

Data Types: `char` | `cell` | `string`

### listener — Listener event handler

function handle

Listener event handler, specified as a function handle to listen for RavenPack News Analytics data events. You can modify the existing listener function `rpExampleListener` or create your own to add other functionality. You can find the code for the existing listener function in the `rpExampleListener.m` file.

Data Types: `function_handle`

## Output Arguments

### `status` — Subscription status

logical

Subscription status, returned as a logical `true` or `false`. `true` or `1` signifies a successful subscription to real-time RavenPack News Analytics data. `false` or `0` signifies an unsuccessful subscription to the real-time data or a subscription that is currently running.

### `lhandle` — RavenPack News Analytics Gateway event listener

handle

RavenPack News Analytics Gateway event listener, returned as a handle to the RavenPack listener object.

## Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

## See Also

`close` | `entitlements` | `ravenpack` | `timeseries`

### Topics

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21
“Workflow for RavenPack News Analytics” on page 3-24

### External Websites

RavenPack Developer Zone Overview

### Introduced in R2015b

# rploader

RavenPack News Analytics file reader

## Syntax

```
d = rploader(filename)
d = rploader(filename,Name,Value)
[d,h] = rploader( ___ )
```

## Description

`d = rploader(filename)` reads the contents of the RavenPack News Analytics data file specified by `filename` and returns the contents in the MATLAB variable `d`.

`d = rploader(filename,Name,Value)` reads the contents of the file using additional options specified by one or more `Name,Value` pair arguments.

`[d,h] = rploader( ___ )` reads the contents of the file and retrieves the header information `h` in the file using any of the input arguments in the previous syntaxes.

## Examples

### Read Data in a File

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data.

```
filename = '2014-11-macro.csv';

d = rploader(filename);
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)

ans =

       TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    ____________________    ____________    ___________    ___________

    24-Nov-2014 00:00:05    'F33A73'        'CMDT'         'Iron Ore'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about Iron Ore.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

### Read Data Using a Name-Value Pair Argument

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. The RavenPack News Analytics entity name is Iron Ore.

```
filename = '2014-11-macro.csv';
```

```
d = rploader(filename,'entity_name',{'Iron Ore'});
```

`d` is a table that contains the RavenPack News Analytics data for Iron Ore.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans =

       TIMESTAMP_UTC         RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    _____________________    ____________    ___________    ___________

    24-Nov-2014 00:00:05     'F33A73'        'CMDT'         'Iron Ore'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about Iron Ore.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

### Read Data Using Multiple Name-Value Pair Arguments

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. Read 5000 records in the data file starting at record number 10,000.

```
filename = '2014-11-macro.csv';

d = rploader(filename,'start',10000,'records',5000);
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans =
```

```
      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE        ENTITY_NAME
    ___________________    ____________    ___________    ____________________

    24-Nov-2014 07:58:38    '0037F3'        'ORGA'         'State of Rajasthan'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about the State of Rajasthan.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

### Read Data Within a Date Range

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Read the data from November 24, 2014 through November 25, 2014. Here, the file contains Equities News Analytics data.

```
filename = '2014-11-equities.csv';

d = rploader(filename,'date',{'11/24/2014','11/25/2014'});
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans =

      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE          ENTITY_NAME
    ___________________    ____________    ___________    ____________________________

    24-Nov-2014 00:00:04    '355013'        'COMP'         'Panoramic Resources Ltd.'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about a company.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 35

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'    'COMPANY'

  Columns 36 through 43

    'ISIN'    'CSS'    'NIP'    'PEQ'    'BEE'    'BMQ'    'BAM'    'BCA'

  Columns 44 through 46

    'BER'    'ANL_CHG'    'MCQ'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

### Read Data and Return the Header Information

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. Read 5000 records in the data file starting at record number 10,000.

```
filename = '2014-11-macro.csv';

[d,h] = rploader(filename,'start',10000,'records',5000);
```

`d` is a table that contains the RavenPack News Analytics data.

`h` is a cell array that contains the header information.

Display the header information.

```
h
```

```
h =

    'TIMESTAMP_UTC'
    'RP_ENTITY_ID'
    'ENTITY_TYPE'
    'ENTITY_NAME'
    'POSITION_NAME'
    'RP_POSITION_ID'
    'COUNTRY_CODE'
    'RELEVANCE'
    'TOPIC'
    'GROUP'
    'TYPE'
    'SUB_TYPE'
    'PROPERTY'
    'EVALUATION_METHOD'
    'MATURITY'
    'CATEGORY'
    'ESS'
    'AES'
    'AEV'
    'ENS'
    'ENS_SIMILARITY_GAP'
    'ENS_KEY'
    'ENS_ELAPSED'
    'G_ENS'
    'G_ENS_SIMILARITY_GAP'
    'G_ENS_KEY'
    'G_ENS_ELAPSED'
    'EVENT_SIMILARITY_KEY'
    'NEWS_TYPE'
    'SOURCE'
    'RP_STORY_ID'
    'RP_STORY_EVENT_INDEX'
    'RP_STORY_EVENT_COUNT'
    'PRODUCT_KEY'
```

For details, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

## Input Arguments

### `filename` — RavenPack News Analytics data file

character vector | string scalar

RavenPack News Analytics data file, specified as a character vector or string scalar. To create this file, use the RavenPack Data Feed Tool to export news data into a comma-separated file.

Data Types: `char` | `string`

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'date',{'11/24/2014'}`

**`date` — Date or date range**
cell array | string array

Date or date range, specified as the comma-separated pair consisting of `'date'` and a cell array or string array. If you specify one date, `rploader` returns news data for the day specified by the string scalar, character vector, date number, or `datetime` array in the cell array. To specify a date range, use `'date'` and a string array that contains two string scalars or a cell array that contains two character vectors, date numbers, or `datetime` arrays separated by a comma. The dates in the date range are inclusive.

Example: `'date',{'11/24/2014','11/25/2014'}`

Data Types: `cell` | `string`

**`rp_entity_id` — RavenPack News Analytics entity identifier**
cell array of character vectors | string array

RavenPack News Analytics entity identifier, specified as the comma-separated pair consisting of `'rp_entity_id'` and a cell array of one or more character vectors or string array. The character vectors or string scalars denote the RavenPack entity identifiers. For details about the RavenPack entity identifiers, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Example: `'rp_entity_id',{'F33A73'}`

Data Types: `cell`

**`entity_name` — RavenPack News Analytics entity name**
cell array of character vectors | string array

RavenPack News Analytics entity name, specified as a comma-separated pair consisting of `'entity_name'` and a cell array of one or more character vectors or string array of string scalars that denote RavenPack entity names. For details about the RavenPack entity names, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Example: `'entity_name',{'Iron Ore'}`

Data Types: `cell`

**`start` — Reading offset**
numeric scalar

Reading offset, specified as a numeric scalar to denote the record from which to start reading the RavenPack News Analytics data in the data file.

Example: `'start',100`

Data Types: `double`

**`records` — Number of records to read**
numeric scalar

Number of records to read in the data file, specified as a numeric scalar.

Example: `'records',5000`

Data Types: `double`

## Output Arguments

**`d` — RavenPack News Analytics data**
table | double

RavenPack News Analytics data, returned as a table. If no matching data is found based on the specified name-value pair arguments, `d` returns as an empty double.

**`h` — Header information**
cell array

Header information, returned as a cell array. The header information contains the titles of each column in the returned data `d`.

## See Also

`entitlements` | `ravenpack` | `realtime` | `timeseries`

**Topics**
“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21
“Workflow for RavenPack News Analytics” on page 3-24

**External Websites**
RavenPack Developer Zone Overview

**Introduced in R2015b**

# timeseries

RavenPack News Analytics intraday and historical data

## Syntax

```
d = timeseries(c,symbol,{startdate,enddate})
d = timeseries(c,symbol,{startdate,enddate},fields)
```

## Description

`d = timeseries(c,symbol,{startdate,enddate})` retrieves RavenPack News Analytics intraday or historical data. This function uses the RavenPack News Analytics connection `c`, RavenPack entitled symbol, and a date range between the start date `startdate` and end date `enddate`.

`d = timeseries(c,symbol,{startdate,enddate},fields)` retrieves RavenPack News Analytics intraday or historical data for specific RavenPack fields.

## Examples

### Retrieve RavenPack News Analytics Intraday Data

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last 15 minutes. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is set to 15 minutes ago. The end date is the current date and time.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-.01;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four variables of the first record of intraday data.

```
d(1,1:4)
```

```
ans =

  1×4 table

      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    ____________________    ____________    ___________    ____________

    14-May-2018 15:07:16       2491BC          COMP        Twitter Inc.
```

Each row in the table is one record of news data. Here, the first four variables specify a news event on May 14, 2018 about a company.

List the variables in the data.

```
d.Properties.VariableNames
```

```
ans =

  1×46 cell array

  Columns 1 through 5

    {'TIMESTAMP_UTC'}    {'RP_ENTITY_ID'}    {'ENTITY_TYPE'}    {'ENTITY_NAME'}    {'POSITION_NAME'}

  Columns 6 through 11

    {'RP_POSITION_ID'}    {'COUNTRY_CODE'}    {'RELEVANCE'}    {'TOPIC'}    {'GROUP'}    {'TYPE'}

  Columns 12 through 17

    {'SUB_TYPE'}    {'PROPERTY'}    {'EVALUATION_METHOD'}    {'MATURITY'}    {'CATEGORY'}    {'ESS'}

  Columns 18 through 23

    {'AES'}    {'AEV'}    {'ENS'}    {'ENS_SIMILARITY_…'}    {'ENS_KEY'}    {'ENS_ELAPSED'}

  Columns 24 through 28

    {'G_ENS'}    {'G_ENS_SIMILARIT…'}    {'G_ENS_KEY'}    {'G_ENS_ELAPSED'}    {'EVENT_SIMILARIT…'}

  Columns 29 through 33

    {'NEWS_TYPE'}    {'SOURCE'}    {'RP_STORY_ID'}    {'RP_STORY_EVENT_…'}    {'RP_STORY_EVENT_…'}

  Columns 34 through 41

    {'PRODUCT_KEY'}    {'COMPANY'}    {'ISIN'}    {'CSS'}    {'NIP'}    {'PEQ'}    {'BEE'}    {'BMQ'}

  Columns 42 through 46

    {'BAM'}    {'BCA'}    {'BER'}    {'ANL_CHG'}    {'MCQ'}
```

For details about each variable in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Close the RavenPack News Analytics connection.

```
close(c)
```

### Retrieve RavenPack News Analytics Intraday Data with Fields

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last minute using RavenPack fields. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is set to a minute ago. The end date is the current date and time. The fields list contains these fields:

- `ENTITY_NAME`
- `CATEGORY`
- `SUB_TYPE`

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-.001;
```

```
enddate = now;
fields = {'ENTITY_NAME','CATEGORY','SUB_TYPE'};

d = timeseries(c,symbol,{startdate,enddate},fields);
```

d is a table that contains RavenPack News Analytics data for the companies with news events. Each row in the table is a news event for a company.

Display the data for the first few news events.

```
head(d)
```

```
ans =

  8×4 table

      TIMESTAMP_UTC              ENTITY_NAME                    CATEGORY       SUB_TYPE
    ____________________    ____________________________    ____________    ____________

    14-May-2018 15:24:43    Canterbury Park Holding Corp.    earnings        <undefined>
    14-May-2018 15:24:45    Petróleos de Venezuela S.A       <undefined>     <undefined>
    14-May-2018 15:24:45    ConocoPhillips Co.               <undefined>     <undefined>
    14-May-2018 15:24:45    Facebook Inc.                    <undefined>     <undefined>
    14-May-2018 15:24:45    Rockwell Automation Inc.         <undefined>     <undefined>
    14-May-2018 15:24:45    ABB Ltd.                         <undefined>     <undefined>
    14-May-2018 15:24:45    Duravant LLC                     <undefined>     <undefined>
    14-May-2018 15:24:45    Honeywell International Inc.     <undefined>     <undefined>
```

Close the RavenPack News Analytics connection.

```
close(c)
```

### Retrieve RavenPack News Analytics Historical Data

Create a RavenPack News Analytics connection c using the user name username and password pwd.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last day. Here, the symbol is set to the entitled symbol (entity-scores :rpna-4.0-eqt). The start date is set to 1 day ago. The end date is the current date and time.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-1;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

d is a table that contains the RavenPack News Analytics data.

To retrieve more than 3 days of historical news data, use the RavenPack News Analytics Data Feed Tool.

Display the first four variables of the first record of historical data.

```
d(1,1:4)
```

```
ans =

  1×4 table

      TIMESTAMP_UTC        RP_ENTITY_ID    ENTITY_TYPE     ENTITY_NAME
    ____________________    ____________    ___________    _____________

    13-May-2018 15:30:32       12E454          COMP        Facebook Inc.
```

Each row in the table is one record of news data. Here, the first four variables specify a news event on May 13, 2018 about a company.

List the variables in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 9

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'    'TOPIC'

  Columns 10 through 14

    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'    'EVALUATION_METHOD'

  Columns 15 through 20

    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'    'ENS'

  Columns 21 through 24

    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 28

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'    'EVENT_SIMILARITY...'

  Columns 29 through 32

    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'    'RP_STORY_EVENT_I...'

  Columns 33 through 38

    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'    'COMPANY'    'ISIN'    'CSS'    'NIP'

  Columns 39 through 46

    'PEQ'    'BEE'    'BMQ'    'BAM'    'BCA'    'BER'    'ANL_CHG'    'MCQ'
```

For details about each variable in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Close the RavenPack News Analytics connection.

```
close(c)
```

### Retrieve RavenPack News Analytics Intraday Data as Timetable

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Set the data return format to timetable using the `DataReturnFormat` property of the `ravenpack` object.

```
c.DataReturnFormat = 'timetable';
```

Retrieve RavenPack News Analytics data for the last 15 minutes. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is set to 15 minutes ago. The end date is the current date and time. `d` is a timetable that contains the RavenPack News Analytics data.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-.01;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

Display the first three variables of the first record of intraday data.

```
d(1,1:3)
```

```
ans =

  1×3 timetable

      TIMESTAMP_UTC         RP_ENTITY_ID    ENTITY_TYPE     ENTITY_NAME
    ____________________    ____________    ___________    _____________

    14-May-2018 15:04:15        12E454          COMP        Facebook Inc.
```

Each row in the timetable is one record of news data. Here, the first three variables specify a news event on May 14, 2018 about a company.

List the variables in the data.

```
d.Properties.VariableNames
```

```
ans =

  1×45 cell array

  Columns 1 through 5

    {'RP_ENTITY_ID'}    {'ENTITY_TYPE'}    {'ENTITY_NAME'}    {'POSITION_NAME'}    {'RP_POSITION_ID'}

  Columns 6 through 11

    {'COUNTRY_CODE'}    {'RELEVANCE'}    {'TOPIC'}    {'GROUP'}    {'TYPE'}    {'SUB_TYPE'}

  Columns 12 through 17

    {'PROPERTY'}    {'EVALUATION_METHOD'}    {'MATURITY'}    {'CATEGORY'}    {'ESS'}    {'AES'}

  Columns 18 through 23

    {'AEV'}    {'ENS'}    {'ENS_SIMILARITY_…'}    {'ENS_KEY'}    {'ENS_ELAPSED'}    {'G_ENS'}

  Columns 24 through 27

    {'G_ENS_SIMILARIT…'}    {'G_ENS_KEY'}    {'G_ENS_ELAPSED'}    {'EVENT_SIMILARIT…'}

  Columns 28 through 32

    {'NEWS_TYPE'}    {'SOURCE'}    {'RP_STORY_ID'}    {'RP_STORY_EVENT_…'}    {'RP_STORY_EVENT_…'}

  Columns 33 through 40

    {'PRODUCT_KEY'}    {'COMPANY'}    {'ISIN'}    {'CSS'}    {'NIP'}    {'PEQ'}    {'BEE'}    {'BMQ'}

  Columns 41 through 45

    {'BAM'}    {'BCA'}    {'BER'}    {'ANL_CHG'}    {'MCQ'}
```

For details about each variable in the timetable, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Close the RavenPack News Analytics connection.

```
close(c)
```

## Input Arguments

### `c` — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

### `symbol` — RavenPack entitled symbol

character vector | string scalar | cell array of character vectors

RavenPack entitled symbol, specified as a character vector, string scalar, or cell array with one character vector.

Data Types: `char` | `cell` | `string`

### `startdate` — Start date

numeric scalar | character vector | string scalar | `datetime` array

Start date, specified as a numeric scalar, character vector, string scalar, or `datetime` array to denote the start date of the date range for the returned news data.

Example: `now-.01`

Data Types: `double` | `char` | `string` | `datetime`

### `enddate` — End date

numeric scalar | character vector | string scalar | `datetime` array

End date, specified as a numeric scalar, character vector, string scalar, or `datetime` array to denote the end date of the date range for the returned news data.

Example: `now`

Data Types: `double` | `char` | `string` | `datetime`

### `fields` — RavenPack field list

character vector | string scalar | cell array of character vectors | string array

RavenPack fields list, specified as a character vector, string scalar, cell array of one or more character vectors, or string array. Each character vector corresponds to a RavenPack field. The fields determine the news data to return. For details about the fields, contact RavenPack.

Example: `{'ENTITY_NAME','CATEGORY','SUB_TYPE'}`

Data Types: `char` | `cell` | `string`

## Output Arguments

### `d` — RavenPack News Analytics data

table

RavenPack News Analytics data, specified as a table.

## Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.
- If you encounter this error, decrease the date range using the input arguments `startdate` and `enddate`.

```
Java exception occurred:
com.ravenpack.data.DataGatewayException: TIMEOUT
while invoking remote function:
jl-api.getQuoteRange
```

## See Also

`close` | `entitlements` | `ravenpack` | `realtime`

**Topics**

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21
“Workflow for RavenPack News Analytics” on page 3-24

**External Websites**

RavenPack Developer Zone Overview

**Introduced in R2015b**

# close

Close RavenPack News Analytics connection

## Syntax

```
close(c)
```

## Description

`close(c)` closes the RavenPack News Analytics connection `c`.

## Examples

### Close the RavenPack News Analytics Connection

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Close the RavenPack News Analytics connection.

```
close(c)
```

## Input Arguments

### c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

## Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

## See Also

`ravenpack`

### Topics

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-21
“Workflow for RavenPack News Analytics” on page 3-24

### External Websites

RavenPack Developer Zone Overview

**Introduced in R2015b**

# elektron

Elektron from Refinitiv Message API connection

## Description

The `elektron` function creates an `elektron` object. The `elektron` object represents a Elektron from Refinitiv connection.

After you create an `elektron` object, you can use the object functions to retrieve current and real-time data. You can retrieve data based on your credentials, which consist of a user name and custom IP address. For credentials, contact Elektron from Refinitiv.

When you install Elektron on your computer, the installation folder contains JAR files. Add these JAR files to the dynamic Java class path every time you connect to Elektron:

- `ansipage.jar`
- `ema-javadoc.jar`
- `ema.jar`
- `jdacsUpalib.jar`
- `upa.jar`
- `upaValueAdd.jar`
- `upaValueAddCache.jar`
- `commons-configuration-1.10.jar`
- `commons-lang-2.6.jar`
- `commons-logging-1.2.jar`
- `org.apache.commons.collections.jar`
- `slf4j-api-1.7.12.jar`
- `slf4j-jdk14-1.7.12.jar`

Alternatively, you can add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

## Creation

### Syntax

```
c = elektron(username,service)
c = elektron(username,service,ipaddress)
c = elektron(username,service,ipaddress,port)
```

#### Description

`c = elektron(username,service)` creates a Elektron connection using a user name and service name.

`c = elektron(username,service,ipaddress)` uses an IP address for the Elektron connection.

`c = elektron(username,service,ipaddress,port)` also sets the Port property.

**Input Arguments**

**`username` — Refinitiv user name**
character vector | string scalar

Refinitiv user name, specified as a character vector or string scalar. To find your user name, contact Refinitiv.

Example: `'username'`

Data Types: `char` | `string`

**`service` — Refinitiv service name**
character vector | string scalar

Refinitiv service name, specified as a character vector or string scalar. This input argument indicates the enabled Elektron service for your Refinitiv user name. To find your service name, contact Refinitiv.

Example: `'servicename'`

Data Types: `char` | `string`

**`ipaddress` — IP address for Refinitiv server**
`'localhost'` (default) | character vector | string scalar

IP address for the Refinitiv server where Elektron data is available, specified as a character vector or string scalar. To find the IP address for the Refinitiv server, contact Refinitiv.

Example: `'123.123.123.123'`

Data Types: `char` | `string`

## Properties

**`Username` — Refinitiv user name**
character vector

Refinitiv user name, specified as a character vector. To find your user name, contact Refinitiv.

The `elektron` function sets this property using the `username` input argument.

Example: `'username'`

Data Types: `char`

**`Service` — Refinitiv service name**
character vector

Refinitiv service name, specified as a character vector. This property indicates the enabled Elektron service for your Refinitiv user name. To find your service name, contact Refinitiv.

The `elektron` function sets this property using the `service` input argument.

Example: `'servicename'`

Data Types: `char`

**`IPAddress` — IP address for Refinitiv server**
`'localhost'` (default) | character vector

IP address for the Refinitiv server where Elektron data is available, specified as a character vector. To find the IP address for the Refinitiv server, contact Refinitiv.

The `elektron` function sets this property using the `ipaddress` input argument.

Example: `'123.123.123.123'`

Data Types: `char`

**`Port` — Port number**
`14002` (default) | numeric scalar

Port number for the Elektron connection, specified as a numeric scalar. To find the port number, contact Refinitiv.

Data Types: `double`

## Object Functions

| | |
|---|---|
| close | Close Elektron from Refinitiv connection |
| getdata | Elektron from Refinitiv current market data |
| realtime | Elektron from Refinitiv real-time market data |

## Examples

### Create Elektron Connection

Create a Elektron connection. Then, retrieve current market data. The current market data you see when completing this example can differ from the output data shown.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using a user name and service name. `c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)

 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type of the Elektron field
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)

d =

  284×4 table array

    FieldId    DataType        Name                Value
    _______    ________    _____________    ________________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)

ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Elektron connection.

```
close(c)
```

### Create Elektron Connection Using IP Address

Create a Elektron connection using an IP address. Then, retrieve current market data. The current market data you see when completing this example can differ from the output data shown.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using a user name, service name, and IP address. `c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
ipaddress = '123.123.123.123';

c = elektron(username,servicename,ipaddress)
```

```
 c =

 elektron with properties:
    IPAddress: '123.123.123.123'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array
```

```
    FieldId    DataType        Name                Value
    _______    ________    _____________    ________________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Elektron connection.

```
close(c)
```

### Create Elektron Connection Using Port Number

Create a Elektron connection using a port number. Then, retrieve current market data. The current market data you see when completing this example can differ from the output data shown.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using a user name, service name, IP address, and port number. `c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
ipaddress = '123.123.123.123';
port = '1234';
```

```
c = elektron(username,servicename,ipaddress,port)
```

```
 c =

 elektron with properties:
    IPAddress: '123.123.123.123'
         Port: 1234
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array

    FieldId    DataType        Name                Value
    _______    ________    _____________    ____________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Elektron connection.

```
close(c)
```

## See Also

**Topics**

“Decide to Buy Shares Using Elektron Current Data” on page 6-2
“Decide to Buy Shares Using Elektron Real-Time Data” on page 6-4

**External Websites**
Elektron from Refinitiv

**Introduced in R2017a**

# getdata

Elektron from Refinitiv current market data

## Syntax

```
d = getdata(c,s)
```

## Description

`d = getdata(c,s)` returns Elektron from Refinitiv current market data using the Elektron connection for a single security.

## Examples

### Retrieve Current Elektron Data

Create a Elektron connection. Then, retrieve current market data. The current market data you see when completing this example can differ from the output data shown.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using a user name and service name. `c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)
```

```
 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
```

```
     Service: 'servicename'
    Username: 'username'
```

Retrieve current data for the IBM security using the Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type of the Elektron field
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array

    FieldId    DataType        Name                Value
    _______    ________    _____________    ____________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Elektron connection.

```
close(c)
```

## Input Arguments

### c — Elektron connection

`elektron` object

Elektron connection, specified as an `elektron` object created using the `elektron` function.

### s — Elektron security

character vector | string scalar

Elektron security, specified as a character vector or string scalar. You can specify only a single security. The security is a Reuters Instrument Code (RIC). For details about RICs, contact Refinitiv.

Example: `'IBM.N'`

Data Types: `char` | `string`

## Output Arguments

### d — Elektron current market data

table

Elektron current market data, specified as a table. The `getdata` function returns Elektron current market data for the specified security `s`.

| Variable | Data Type | Description |
|---|---|---|
| `FieldId` | Cell array of doubles | Elektron field identifier |
| `DataType` | Cell array of doubles | Elektron data type of the Elektron field |
| `Name` | Cell array of character vectors | Elektron field name |
| `Value` | Cell array of doubles or character vectors | Current Elektron data value |

MATLAB converts the current data value from Elektron as follows:

- Numeric data values convert to doubles
- Text values convert to character vectors
- Date and time values convert to character vectors

For details about fields and their availability, see Elektron .

## See Also

`close` | `elektron` | `realtime`

### Topics

“Decide to Buy Shares Using Elektron Current Data” on page 6-2

### External Websites

Elektron from Refinitiv

### Introduced in R2017a

# realtime

Elektron from Refinitiv real-time market data

## Syntax

```
reqid = realtime(c,seclist,eventhandler)
reqid = realtime(c,r,eventhandler)
```

## Description

`reqid = realtime(c,seclist,eventhandler)` returns a real-time request identifier using the Elektron from Refinitiv connection, security list, and an event handler function. You can retrieve Elektron real-time market data by accessing variables that appear in the MATLAB workspace.

`reqid = realtime(c,r,eventhandler)` retrieves Elektron real-time market data using a Elektron custom request message.

## Examples

### Retrieve Elektron Real-Time Data

First, create a Elektron connection. Then, retrieve real-time market data. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
```

```
c = elektron(username,servicename);
```

Retrieve real-time market data for the IBM security using the Elektron connection. Use the example event handler function `elektronExampleListener`. To access the code for this function, enter `edit elektronExampleListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
seclist = 'IBM.N';
eventhandler = @(~,ev)elektronExampleListener(ev);
reqid = realtime(c,seclist,eventhandler)
```

```
reqid =

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The cell array `IBM` appears in the MATLAB workspace. `IBM` contains four columns. The columns are:

- Elektron field identifier
- Elektron field name
- Elektron field data type
- Elektron field real-time data value

Access real-time data in the cell array. Access the Elektron field name in the second column and the current data value in the fourth column.

```
field = IBM{1,2};
value = IBM{1,4};
```

For events that do not have an event name or that are not associated with the security, the `ElektronExampleData` object appears in the MATLAB workspace. This object contains decoded event data. View the contents of this object. The contents of this object vary depending on the Elektron event.

`ElektronExampleData`

```
ElektronExampleData =

UpdateMsg
    streamId="10"
    domain="MarketPrice Domain"
    updateTypeNum="1"
    Payload dataType="FieldList"
        FieldList
            FieldEntry fid="22" name="BID" dataType="Real" value="57.1"
            FieldEntry fid="25" name="ASK" dataType="Real" value="57.11"
            FieldEntry fid="30" name="BIDSIZE" dataType="Real" value="22.0"
            FieldEntry fid="31" name="ASKSIZE" dataType="Real" value="39.0"
```

```
            FieldEntry fid="11683" name="BIDFINMMID" dataType="Rmtes" value="(blank data)"
            FieldEntry fid="11684" name="ASKFINMMID" dataType="Rmtes" value="(blank data)"
            FieldEntry fid="3298" name="BIDXID" dataType="Enum" value="43"
            FieldEntry fid="3297" name="ASKXID" dataType="Enum" value="6"
            FieldEntry fid="6579" name="BID_COND_N" dataType="Rmtes" value="R"
            FieldEntry fid="6580" name="ASK_COND_N" dataType="Rmtes" value="R"
            FieldEntry fid="293" name="BID_MMID1" dataType="Rmtes" value="NAS"
            FieldEntry fid="296" name="ASK_MMID1" dataType="Rmtes" value="XPH"
            FieldEntry fid="1000" name="GV1_TEXT" dataType="Rmtes" value="A"
            FieldEntry fid="8937" name="LIMIT_INDQ" dataType="Enum" value="25"
            FieldEntry fid="3887" name="SEQNUM_QT" dataType="Real" value="1.6923329E7"
            FieldEntry fid="118" name="PRC_QL_CD" dataType="Enum" value="0"
            FieldEntry fid="3264" name="PRC_QL3" dataType="Enum" value="0"
            FieldEntry fid="8406" name="QTE_ORIGIN" dataType="Rmtes" value=" "
            FieldEntry fid="1041" name="GV1_FLAG" dataType="Rmtes" value=" "
            FieldEntry fid="12783" name="NBBO_IND" dataType="Enum" value="5"
            FieldEntry fid="3855" name="QUOTIM_MS" dataType="UInt" value="62664591"
            FieldEntry fid="1025" name="QUOTIM" dataType="Time" value="17:24:24:000:000:000"
            FieldEntry fid="14238" name="ORDRECV_MS" dataType="Time" value="17:24:24:590:000:000"
            FieldEntry fid="14246" name="ORDREC2_MS" dataType="Time" value="(blank data)"
        FieldListEnd
    PayloadEnd
UpdateMsgEnd
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Elektron connection.

```
close(c)
```

### Retrieve Elektron Real-Time Last Trade Price Data

First, create a Elektron connection. Then, retrieve real-time last trade price data. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
```

```
c = elektron(username,servicename);
```

Retrieve real-time market data for the IBM security using the Elektron connection. Use the example event handler function `elektronPriceTableListener`. To access the code for this function, enter `edit elektronPriceTableListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
seclist = 'IBM.N';
eventhandler = @(~,ev)elektronPriceTableListener(ev,seclist);
reqid = realtime(c,seclist,eventhandler)
```

```
reqid =

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The table `PriceTable` appears in the MATLAB workspace. `PriceTable` contains these variables:

- `RIC` — RIC for the IBM security
- `TRDPRC_1` — Last trade price data

Elektron continuously updates the `TRDPRC_1` variable with the latest trade price in the table row.

Access the last trade price in real time for the IBM security.

```
PriceTable.TRDPRC_1
```

```
ans =

  163.2600
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Elektron connection.

```
close(c)
```

### Retrieve Elektron Real-Time Data Using Custom Request Message

First, create a Elektron connection. Then, retrieve real-time data using a custom request message. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

Create a custom request message for the Apple security. Enter this Elektron Message API code.

```
import com.thomsonreuters.ema.access.*;
import com.thomsonreuters.ema.access.OmmConsumerConfig.*;

r = EmaFactory.createReqMsg;
r.serviceName(c.Service).name('AAPL.O');
```

Retrieve last trade price data using the Elektron connection and custom request message. Use the example event handler function `elektronExampleListener`. To access the code for this function, enter `edit elektronExampleListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
eventhandler = @(~,ev)elektronExampleListener(ev);
reqid = realtime(c,r,eventhandler)
```

```
reqid =

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
```

```
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The cell array AAPL appears in the MATLAB workspace.

Access real-time data in the cell array. Access the Elektron field name in the second column and the current data value in the fourth column.

```
field = AAPL{1,2};
value = AAPL{1,4};
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Elektron connection.

```
close(c)
```

## Input Arguments

### `c` — Elektron connection

`elektron` object

Elektron connection, specified as an `elektron` object created using the `elektron` function.

### `seclist` — Elektron security list

character vector | string scalar | cell array of character vectors | string array

Elektron security list, specified as a character vector, string scalar, cell array of character vectors, or string array. Specify securities in the security list as:

- A character vector or string for one security.
- A cell array of character vectors or a string array for multiple securities. Each character vector in the cell array or each string in the string array is a Elektron security.

The security is a Reuters Instrument Code (RIC). For details about RICs, contact Refinitiv.

Example: `{'IBM.N','MSFT.O'}`

Data Types: `cell` | `char` | `string`

### `r` — Elektron request message

Elektron Message API request object

Elektron request message, specified as a Elektron Message API request object. To create this object, use the Elektron Message API code. For details, see Elektron from Refinitiv. This object defines the data and parameters for a custom message request.

### `eventhandler` — Event handler

function handle

Event handler, specified as a function handle. You can use the example event handling functions `elektronExampleListener` or `elektronPriceTableListener` to process real-time Elektron Message API events. Or, you can define a custom event handler function to process events of your choice. For details, see “Writing and Running Custom Event Handler Functions” on page 1-26.

The event handler function `elektronExampleListener` creates a cell array for each security in the input argument `seclist`. The cell array contains real-time data for the security. For example, if `seclist` contains `'IBM.N'`, then the cell array named `IBM` appears in the MATLAB workspace.

For events without a name or that are not associated with a RIC, the event handler function creates an object named `ElektronExampleData` in the MATLAB workspace. This object contains decoded data for these events. The object type differs depending on the current Elektron Message API output. For details about this object, see Elektron from Refinitiv.

This table describes the columns in the cell arrays for each security that appear in the MATLAB workspace.

| Cell Array Column | Description |
|---|---|
| First column | Elektron field identifier |
| Second column | Elektron field name |
| Third column | Elektron data type of the Elektron field |
| Fourth column | Current Elektron data value |

The event handler function `elektronExampleListener` has an input argument `ev` that is specified as a four-column cell array that contains:

- Decoded data
- Elektron Message API event object
- Event message
- Event name

The other event handler function `elektronPriceTableListener` creates a table named `PriceTable` in the MATLAB workspace. This table contains the last trade price for each security in the input argument `seclist`. The table contains two variables, `RIC` for the security and `TRDPRC_1` for the last trade price of the corresponding security:

- `RIC` is returned as a cell array of one or more character vectors that depend on the number of securities in the security list.
- `TRDPRC_1` is returned as a numeric scalar for one security or an array of doubles that has a value for each security in `RIC`.

The event handler function `elektronPriceTableListener` has the security list as an additional input argument. The security list matches the contents of `seclist`.

To access the code for the event handler function `elektronExampleListener`, enter `edit elektronExampleListener.m`. To access the code for the event handler function `elektronPriceTableListener`, enter `edit elektronPriceTableListener.m`.

For example, to retrieve real-time data for the IBM and Microsoft securities, enter this code that assumes an established Elektron connection `c`.

```
seclist = {'IBM.N','MSFT.O'};
reqId = realtime(c,seclist,@(~,ev)elektronExampleListener(ev));
```

Example: `@(~,ev)elektronExampleListener(ev)`

Data Types: `function_handle`

## Output Arguments

### `reqid` — Real-time request identifier

structure

Real-time request identifier, returned as a structure with these fields.

| Field | Description |
|---|---|
| `ReqId` | Request identifier for the real-time data request |
| `ReqMsg` | Elektron Message API request object |
| `Handle` | MATLAB event listener process object |
| `Listener` | MATLAB event listener object |

For details about the request identifier and request object, see Elektron from Refinitiv.

For details about MATLAB event listeners, see “Overview Events and Listeners”.

Use the real-time request identifier to stop the real-time data subscription; for example:

```
delete(reqid.Listener)
```

## Tips

- If you encounter issues with Elektron data retrieval, see the contents of the `ElektronExampleData` object in the MATLAB workspace. For example, if you enter an invalid RIC, the `ElektronExampleData` object contains a status message that specifies the RIC cannot be found. To retrieve such information automatically, write a custom event handler function that parses status messages in the `ElektronExampleData` object. For details about custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-26. For details about the status messages, see Elektron from Refinitiv.

## See Also

`close` | `elektron` | `getdata`

**Topics**

“Decide to Buy Shares Using Elektron Real-Time Data” on page 6-4
“Writing and Running Custom Event Handler Functions” on page 1-26

**External Websites**

Elektron from Refinitiv

**Introduced in R2017a**

# close

Close Elektron from Refinitiv connection

## Syntax

```
close(c)
```

## Description

`close(c)` closes the Elektron from Refinitiv connection.

## Examples

### Close Elektron Connection

Create a Elektron connection. Then, retrieve current market data. The current market data you see when completing this example can differ from the output data shown.

Add Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path”.

Connect to Elektron using a user name and service name. `c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)
```

```
 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
```

```
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Elektron field identifier
- `DataType` — Elektron data type of the Elektron field
- `Name` — Elektron field name
- `Value` — Current Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array

    FieldId    DataType        Name                Value
    _______    ________    _____________    ________________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Elektron connection.

```
close(c)
```

## Input Arguments

### c — Elektron connection

`elektron` object

Elektron connection, specified as an `elektron` object created using the `elektron` function.

## See Also

`elektron` | `getdata` | `realtime`

### Topics

“Decide to Buy Shares Using Elektron Current Data” on page 6-2

“Decide to Buy Shares Using Elektron Real-Time Data” on page 6-4

**External Websites**
Elektron from Refinitiv

**Introduced in R2017a**

# rdth

(To be removed) Connect to Thomson Reuters Tick History

**Note** The `rdth` function will be removed in a future release. Use the `trth` function instead. For details, see “Compatibility Considerations”.

## Syntax

```
r = rdth(username,password)
r = rdth(username,password,[],flag)
```

## Description

`r = rdth(username,password)` creates a Thomson Reuters Tick History connection to enable intraday tick data retrieval.

`r = rdth(username,password,[],flag)` sets the reference data flag `flag` to toggle the return of reference data.

## Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```

returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

Suppose that you want to get the intraday price and volume information for all ticks of type Trade. To determine which fields apply to the message type `Trade` and the `requestType` of the Trade message, the command:

```
v = get(r,'MessageTypes')
```

returns

```
v = RequestType: {31x1 cell}
Name: {31x1 cell}
Fields: {31x1 cell}
```

The command

```
v.Name
```

then returns

```
ans =
    'C&E Quote'
    'Short Sale'
    'Fund Stats'
    'Economic Indicator'
    'Convertibles Transactions'
    'FI Quote'
    'Dividend'
    'Trade'
    'Stock Split'
    'Settlement Price'
    'Index'
    'Open Interest'
    'Correction'
    'Quote'
    'OTC Quote'
    'Stock Split'
    'Market Depth'
    'Dividend'
    'Stock Split'
    'Market Maker'
    'Dividend'
    'Stock Split'
    'Intraday 1Sec'
    'Dividend'
    'Intraday 5Min'
    'Intraday 1Min'
    'Intraday 10Min'
    'Intraday 1Hour'
    'Stock Split'
    'End Of Day'
    'Dividend'
```

The command

```
j = find(strcmp(v.Name,'Trade'));
```

returns

```
j =     8
```

The command

```
v.Name{j}
```

returns

```
ans = Trade
```

The command

```
v.RequestType{8}
```

returns

```
ans = TimeAndSales
```

The command

```
v.Fields{j}
```

returns

```
ans =
    'Exchange ID'
    'Price'
    'Volume'
    'Market VWAP'
    'Accumulative Volume'
    'Turnover'
    'Buyer ID'
    'Seller ID'
    'Qualifiers'
    'Sequence Number'
    'Exchange Time'
    'Block Trade'
    'Floor Trade'
    'PE Ratio'
    'Yield'
    'Implied Volatility'
    'Trade Date'
    'Tick Direction'
    'Dividend Code'
    'Adjusted Close Price'
    'Price Trade-Through-Exempt Flag'
    'Irregular Trade-Through-Exempt Flag'
    'TRF Price Sub Market ID'
    'TRF'
    'Irregular Price Sub Market ID'
```

To request the Exchange ID, Price, and Volume of a security's intraday tick for a given day and time range the command

```
x = fetch(r,'ABCD.O',{'Exchange ID','Price','Volume'},...
{'09/05/2008 12:00:06','09/05/2008 12:00:10'},...
'TimeAndSales','Trade','NSQ','EQU');
```

returns data similar to

```
x =

    'ABCD.O'  '05-SEP-2008'  '12:00:08.535' ...
   'Trade'    'NAS'    '85.25'    '100'
    'ABCD.O'  '05-SEP-2008'  '12:00:08.569' ...
   'Trade'    'NAS'    '85.25'    '400'
```

To request the Exchange ID, Price, and Volume of a security's intraday tick data for an entire trading day, the command

```
x = fetch(r,'ABCD.O',{'Exchange ID','Price','Volume'},...
'09/05/2008','TimeAndSales','Trade','NSQ','EQU');
```

returns data similar to

```
x =

    'ABCD.O'    '05-SEP-2008'    '08:00:41.142'...
    'Trade'    'NAS'    '51'       '100'
    'ABCD.O'    '05-SEP-2008'    '08:01:03.024'...
    'Trade'    'NAS'    '49.35'    '100'
```

```
    'ABCD.O'    '05-SEP-2008'    '19:37:47.934'...
    'Trade'    'NAS'    '47.5'    '1200'
    'ABCD.O'    '05-SEP-2008'    '19:37:47.934'...
    'Trade'    'NAS'    '47.5'    '300'
    'ABCD.O'    '05-SEP-2008'    '19:59:33.970'...
    'Trade'    'NAS'    '47'      '173'
```

To clean up any remaining requests associated with the rdth connection use:

```
close(r)
```

To create a Thomson Reuters Tick History connection so that subsequent data requests do not return reference data, use:

```
 r = rdth('user@company.com','mypassword',[],false)
```

returns

```
r =
        client: [1x1 com.thomsonreuters.tickhistory.webservice.TRTHApiServiceStub]
          user: 'user@company.com'
      password: '**********'
          cred: [1x1 com.thomsonreuters.tickhistory.webservice.types.CredentialsHeaderE]
   refDataFlag: 0
```

The property flag can be modified after making the connection with:

```
r.refDataFlag = true
```

or

```
r.refDataFlag = false
```

To clean up any remaining requests associated with the rdth connection use:

```
close(r)
```

## Compatibility Considerations

### `rdth` function will be removed

*Not recommended starting in R2017b*

The `rdth` function will be removed in a future release. Use the `trth` function instead. Some differences between the workflows require updates to your code.

#### Update Code

In prior releases, you created a connection to Thomson Reuters Tick History by using the `rdth` function. For example:

```
username = 'user@company.com';
password = 'mypassword';
r = rdth(username,password);
```

Now use the `trth` function instead.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

## See Also

`close` | `fetch` | `get`

**Introduced in R2009a**

# close

(To be removed) Close Thomson Reuters Tick History connection

**Note** The `close` function will be removed in a future release with no replacement.

## Syntax

```
close(r)
```

## Description

`close(r)` closes the Thomson Reuters Tick History connection, `r`.

## See Also

`rdth`

**Introduced in R2009a**

# fetch

(To be removed) Request Thomson Reuters Tick History data

**Note** The `fetch` function will be removed in a future release. For details, see “Compatibility Considerations”.

## Syntax

```
x = fetch(r,sec)
x = fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain)
x =
fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain,marketdept
h)
```

## Description

`x = fetch(r,sec)` returns information about the security, `sec`, such as the code, currency, exchange, and name. `r` is the Thomson Reuters Tick History connection object.

`x = fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain)` returns data for the request security `sec`, based on the type request `reqtype` and message types `messtype`. The value of `messtype` is a cell array of character vectors or string array. Data for the fields specified by `tradefields` is returned for the data range bounded by `daterange`. The value of `tradefields` is a cell array of multiple cell arrays of character vectors, string scalars, or string arrays. Each item in the `tradefields` cell array denotes a unique field list for each message type. Specifying the `exchange` of the given security improves the speed of the data request. `domain` specifies the security type.

`x = fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain,marketdepth)` additionally specifies the depth of level 2 data, `marketdepth`, to return for a `'MarketDepth'` request type. `marketdepth` must be a numeric value between `1` and `10`, returning up to 10 bid/ask values for a given security.

**Note** Do not use date ranges for end of day requests. You can specify a range of hours on a single day, but not a multiple day range.

## Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```

returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
```

```
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get information pertaining to a particular security, the command

```
d = fetch(r,'GOOG.O',{'Volume','Price','Exchange ID'},...
{'09/05/2008 12:00:00','09/05/2008 12:01:00'},...
'TimeAndSales','Trade','NSQ','EQU')
```

returns data starting with (not all data is shown):

```
d =
'#RIC'      'Date[L]'        'Time[L]'         'Type'...
     'Ex/Cntrb.ID'    'Price'
'GOOG.O'    '05-SEP-2008'    '12:00:01.178'    'Trade'...
    'NAS'            '443.86'
'Volume'
'200'
```

The command

```
d = fetch(r,'GOOG.O',{'Volume','Last'},{'09/05/2008'},...
'EndOfDay','End Of Day','NSQ','EQU')
```

returns

```
d =
    '#RIC'      'Date[L]'        'Time[L]'       ...
  'Type'    'Last'     'Volume'
    'GOOG.O'    '05-SEP-2008'    '23:59:00.000'   ...
 'End Of Day'    '444.25'    '4538375'
```

For

```
x = fetch(r,'GOOG.O')
```

for example, the exchange of the security is `x.Exchange` or `NSQ`. To determine the asset domain of the security, use the value of `x.Type`, in this case 113. Using the information from `v = get(r)`,

```
j = find(v.InstrumentTypes.Value == 113)
```

returns

```
j =46
```

The command

```
v.InstrumentTypes.Value(j)
```

returns

```
ans =
   113
```

The command

```
v.InstrumentTypes.Name(j)
```

returns

```
ans =
    'Equities'
```

The command

```
v.AssetDomains.Value(strcmp(v.InstrumentTypes.Name(j),...
v.AssetDomains.Name))
```

returns

```
ans =
    'EQU'
```

Knowing the security exchange and domain helps the interface to resolve the security symbol and return data more quickly.

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = fetch(R,'AAPL.O',{'Bid Price','Bid Size'},...
        {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

## Tips

- To obtain more information request and message types and their associated field lists, use the command `get(r)`.

## Compatibility Considerations

### `fetch` function will be removed

*Not recommended starting in R2017b*

The `fetch` function will be removed in a future release. Use either the `history` or `timeseries` function instead. Some differences between the workflows might require updates to your code.

#### Update Code

Use the `history` function to retrieve Thomson Reuters Tick History historical data and the `timeseries` function to retrieve intraday data.

In prior releases, you retrieved data using the `fetch` function. For example:

```
r = rdth('user@company.com','mypassword');
d = fetch(r,'GOOG.O',{'Volume','Price','Exchange ID'},...
{'09/05/2008 12:00:00','09/05/2008 12:01:00'},...
'TimeAndSales','Trade','NSQ','EQU');
```

Now use the `history` function to retrieve historical data.

```
username = 'username';
password = 'password';
c = trth(username,password);

sec = ["IBM.N","Ric"];
fields = ["Open";"Last"];
startdate = datetime('yesterday');
enddate = datetime('today');
d = history(c,sec,fields,startdate,enddate);
```

Or, use the `timeseries` function to retrieve intraday data.

```
username = 'username';
password = 'password';
c = trth(username,password);

sec = ["IBM.N","Ric"];
fields = ["Trade - Exchange Time";"Trade - Price";"Trade - Volume"];
startdate = datetime('11/06/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/07/2017','InputFormat','MM/dd/yyyy');
d = timeseries(c,sec,fields,startdate,enddate);
```

## See Also

`close` | `get` | `rdth`

**Introduced in R2009a**

# get

(To be removed) Get Thomson Reuters Tick History connection properties

**Note** `get` will be removed in a future release with no replacement.

## Syntax

```
v = get(r,'propertyname')
v = get(r)
```

## Description

`v = get(r,'propertyname')` returns the value of the specified properties for the `rdth` connection object. `'PropertyName'` is a character vector, string, cell array of character vectors, or string array that contains property names.

`v = get(r)` returns a structure where each field name is the name of a property of `r`, and each field contains the value of that property.

Properties include:

- `AssetDomains`
- `BondTypes`
- `Class`
- `Countries`
- `CreditRatings`
- `Currencies`
- `Exchanges`
- `FuturesDeliveryMonths`
- `InflightStatus`
- `InstrumentTypes`
- `MessageTypes`
- `OptionExpiryMonths`
- `Quota`
- `RestrictedPEs`
- `Version`

## Examples

To create a Thomson Reuters Tick History connection, the command

```
r = rdth('user@company.com','mypassword')
```

returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get a listing of properties for the `rdth` connection, the command

```
v = get(r)
```

returns

```
v =

             AssetDomains: [1x1 struct]
                BondTypes: {255x1 cell}
                    Class: 'class com.thomsonreuters. ...
tickhistory.webservice.client.RDTHApiClient'
                Countries: {142x1 cell}
            CreditRatings: {82x1 cell}
               Currencies: [1x1 struct]
                Exchanges: [1x1 struct]
    FuturesDeliveryMonths: {12x1 cell}
           InflightStatus: [1x1 com.thomsonreuters. ...
tickhistory.webservice.types.InflightStatus]
          InstrumentTypes: [1x1 struct]
             MessageTypes: [1x1 struct]
       OptionExpiryMonths: {12x1 cell}
                    Quota: [1x1 com.thomsonreuters. ...
tickhistory.webservice.types.Quota]
            RestrictedPEs: {2758x1 cell}
                  Version: [1x1 com.thomsonreuters. ...
tickhistory.webservice.types.Version]
```

## See Also

`fetch` | `rdth`

**Introduced in R2009a**

# isconnection

(To be removed) Determine if Thomson Reuters Tick History connections are valid

**Note** `isconnection` will be removed in a future release with no replacement.

## Syntax

```
x = isconnection(r)
```

## Description

`x = isconnection(r)` returns `1` if `r` is a valid `rdth` client and `0` otherwise.

## Examples

Verify that `r` is a valid connection:

```
r = rdth('user@company.com','mypassword');
x = isconnection(r)
x = 1
```

## See Also

`close` | `fetch` | `get` | `rdth`

**Introduced in R2009a**

# status

(To be removed) Status of FTP request for Thomson Reuters Tick History data

**Note** `status` will be removed in a future release with no replacement.

## Syntax

```
[s,qp] = status(r,x)
```

## Description

`[s,qp] = status(r,x)` returns the status and queue position of the Thomson Reuters Tick History (TRTH) FTP request handle, `x`. When `s` is equal to `'Complete'`, download the file from the TRTH server manually or programmatically.

## Examples

Check the status of your FTP request:

```
x = submitftp(r,'GOOG.O',{'Exchange ID','Price','Volume'}, ...
   {(floor(now))-10,(floor(now))},'TimeAndSales','Trade', ...
   'NSQ','EQU')

s = [];
while ~strcmp(s,'Complete')
[s,qp] = status(r,x);
end
```

Optionally, download the file from the TRTH server programmatically. The data file is generated in a directory, `api-results`, on the server. The file has extension `csv.gz`.

```
filename = ['/api-results/' char(x) '-report.csv.gz'];
urlwrite(['https://tickhistory.thomsonreuters.com/HttpPull/Download?'...
          'user=' username '&pass=' password '&file=' filename''],...
          'rdth_results.csv.gz');
```

This call to `urlwrite` saves the downloaded file with the name `rdth_results.csv.gz` in the current directory.

## See Also

`close` | `rdth` | `submitftp`

**Introduced in R2011a**

# submitftp

(To be removed) Submit FTP request for Thomson Reuters Tick History data

**Note** `submitftp` will be removed in a future release with no replacement.

## Syntax

```
x = submitftp(r, sec)
x = submitftp(r, sec, tradefields, daterange, reqtype,
messtype, exchange, domain)
x = submitftp(r,sec,tradefields, daterange, reqtype,
messtype, exchange, domain, marketdepth)
```

## Description

`x = submitftp(r, sec)` returns information about the security, `sec`, such as the code, currency, exchange, and name for the given `trth` connection object, `r`.

`x = submitftp(r, sec, tradefields, daterange, reqtype, messtype, exchange, domain)` submits an FTP request for the request security, `sec`, based on the type request and message types, `reqtype` and `messtype`, respectively. Data for the fields specified by `tradefields` is returned for the data range bounded by `daterange`. Specifying the `exchange` or the given security improves the speed of the data request. `domain` specifies the security type.

`x = submitftp(r,sec,tradefields, daterange, reqtype, messtype, exchange, domain, marketdepth)` additionally specifies the depth of level 2 data, `marketdepth`, to return for a `'MarketDepth'` request type. `marketdepth` must be a numeric value between 1 and 10, returning up to 10 bid/ask values for a given security.

To monitor the status of the FTP request, enter the command

```
[s,qp] = status(r,x)
```

The `status` function returns a status message and queue position. When `S = 'Complete'`, download the resulting compressed `.csv` file from the TRTH servers. Once the `.csv` file has been saved to disk, use `rdthloader('filename')` to load the data into the MATLAB workspace. To obtain more information request and message types and their associated field lists, use the command `get(r)`.

## Examples

Specify parameters for FTP request:

```
submitftp(r,{'IBM.N','GOOG.O'}, ...
   {'Open','Last','Low','High'}, ...
   {floor(now)-100,floor(now)}, ...
   'EndOfDay','End Of Day','NSQ','EQU');
```

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = submitftp(R,'AAPL.O',{'Bid Price','Bid Size'},...
         {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

## See Also

`fetch` | `get` | `rdth` | `rdthloader` | `status`

**Introduced in R2011a**

# rdthloader

(To be removed) Retrieve data from Thomson Reuters Tick History file

**Note** `rdthloader` will be removed in a future release with no replacement.

## Syntax

```
x = rdthloader(file)
x = rdthloader(file,'date',{DATE1})
x = rdthloader(file,'date',{DATE1, DATE2})
x = rdthloader(file,'security',{SECNAME})
x = rdthloader(file,'start',STARTREC)
x = rdthloader(file,'records', NUMRECORDS)
```

## Arguments

Specify the following arguments as name-value pairs. You can specify any combination of name-value pairs in a single call to `rdthloader`.

| | |
|---|---|
| `file` | Thomson Reuters Tick History file from which to retrieve data. |
| `'date'` | Use this argument with `{DATE1,DATE2}` to retrieve data between and including the specified dates. Specify the dates as numbers, character vectors, or string scalars. |
| `'security'` | Use this argument to retrieve data for `SECNAME`, where `SECNAME` is a cell array containing a list of security identifiers for which to retrieve data. |
| `'start'` | Use this argument to retrieve data beginning with the record `STARTREC`, where `STARTREC` is the record at which `rdthloader` begins to retrieve data. Specify `STARTREC` as a number. |
| `'records'` | Use this argument to retrieve `NUMRECORDS` number of records. |

## Description

`x = rdthloader(file)` retrieves tick data from the Thomson Reuters Tick History file `file` and stores it in the structure `x`.

`x = rdthloader(file,'date',{DATE1})` retrieves tick data from `file` with date stamps of value `DATE1`.

`x = rdthloader(file,'date',{DATE1, DATE2})` retrieves tick data from `file` with date stamps between `DATE1` and `DATE2`.

`x = rdthloader(file,'security',{SECNAME})` retrieves tick data from `file` for the securities specified by `SECNAME`.

`x = rdthloader(file,'start',STARTREC)` retrieves tick data from `file` beginning with the record specified by `STARTREC`.

`x = rdthloader(file,'records', NUMRECORDS)` retrieves `NUMRECORDS` number of records from `file`.

## Examples

Retrieve all ticks from the file `'file.csv'` with date stamps of `'02/02/2007'`:

```
x = rdthloader('file.csv','date',{'02/02/2007'})
```

Retrieve all ticks from `'file.csv'` between and including the dates `'02/02/2007'` and `'02/03/2007'`:

```
 x = rdthloader('file.csv','date',{'02/02/2007',...
'02/03/2007'})
```

Retrieve all ticks from `'file.csv'` for the security `'XYZ.O'`:

```
 x = rdthloader('file.csv','security',{'XYZ.O'})
```

Retrieve the first 10,000 tick records from `'file.csv'`:

```
 x = rdthloader('file.csv','records',10000)
```

Retrieve data from `'file.csv'`, starting at record 100,000:

```
  x = rdthloader('file.csv','start',100000)
```

Retrieve up to 100,000 tick records from `'file.csv'`, for the securities `'ABC.N'` and `'XYZ.O'`, with date stamps between and including the dates `'02/02/2007'` and `'02/03/2007'`:

```
x = rdthloader('file.csv','records',100000,...
               'date',{'02/02/2007','02/03/2007'},...
               'security',{'ABC.N','XYZ.O'})
```

## See Also

`reuters` | `rnseloader`

**Introduced in R2008b**

# reuters

Create Reuters sessions

## Description

The `reuters` function creates a `reuters` object. The `reuters` object represents an Enterprise Platform from Refinitiv connection.

After you create a `reuters` object, you can use the object functions to retrieve data from Enterprise Platform and send data to Enterprise Platform.

## Creation

### Syntax

```
c = reuters(session,servicename)
c = reuters(session,servicename,user,position)
c = reuters(session,servicename,[],[],1)
```

#### Description

`c = reuters(session,servicename)` creates a connection to the Enterprise Platform using the session name and sets the serviceName property.

`c = reuters(session,servicename,user,position)` creates a Reuters connection with Data Access Control System (DACS) authentication, and also sets the user and position properties.

`c = reuters(session,servicename,[],[],1)` creates a Reuters connection to access only real-time data from Enterprise Platform.

#### Input Arguments

**`session` — Session name**
character vector | string scalar

Session name, specified as a character vector or string scalar to denote a Reuters session.

Example: `'myNS::remoteSession'`

Data Types: `char` | `string`

### Properties

**`session` — Session name**
Reuters session object

Session name, specified as a Reuters session object to denote the Reuters session.

This `reuters` function sets this property using the `session` input argument.

Example: `[1x1 com.reuters.rfa.internal.session.SessionImpl]`

**`user` — User name**
character vector | string scalar

User name, specified as a character vector or string scalar to denote your Reuters user identification.

Example: `'mw335'`

Data Types: `char` | `string`

**`position` — IP address**
character vector | string scalar

IP address, specified as a character vector or string scalar to identify the machine running the Reuters data server.

Example: `'111.222.333.444/net'`

Data Types: `char` | `string`

**`application` — MATLAB application identifier**
`'182'` (default)

This property is read-only.

MATLAB application identifier, specified as the value `'182'`. Reuters assigns this value to identify the MATLAB application.

**`standardPI` — Reuters standard principal identity**
`Standard Principal Identity` object

Reuters standard principal identity, specified as a Reuters `Standard Principal Identity` object.

Example: `[1x1 com.reuters.rfa.common.StandardPrincipalIdentity]`

**`client` — Reuters client**
client object

Reuters client, specified as a Reuters client object.

Example: `[1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]`

**`serviceName` — Service name**
character vector | string scalar

Service name, specified as a character vector or string scalar that indicates the service for connecting to the Reuters data server.

Example: `'dIDN_RDF'`

Data Types: `char` | `string`

**`eventQueue` — Event queue**
event queue object

Event queue, specified as a Reuters event queue object.

Example: `[90 com.reuters.rfa.internal.common.EventQueueImpl]`

**`marketDataSubscriber` — Event source**
event source object

Event source, specified as a Reuters event source object.

Example: `[1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]`

**`marketDataSubscriberInterestSpec` — Event source interest specification**
event source interest specification object

Event source interest specification, specified as a Reuters event source interest specification object.

Example: `[1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]`

**`mdsClientHandle` — Handle for event stream**
handle object

Handle for the event stream, specified as a Reuters handle object.

Example: `[1x1 com.reuters.rfa.internal.common.HandleImpl]`

**`defDb` — Historical data field list**
field names list object

Historical data field list, specified as a Reuters field names list object.

Example: `[369 com.reuters.ts1.TS1DefDb]`

## Object Functions

## Retrieve Data from Enterprise Platform

| | |
|---|---|
| close | Release connections to Reuters data servers |
| fetch | Request data from Reuters data servers |
| get | Retrieve properties of Reuters session objects |
| history | Request data from Reuters Time Series One |
| stop | Unsubscribe securities |

## Send Data to Enterprise Platform

| | |
|---|---|
| addric | Create Reuters Instrument Code |
| contrib | Contribute data to Reuters data feed |
| deleteric | Delete Reuters Instrument Code |

## Examples

### Connect to Enterprise Platform

Create an Enterprise Platform connection. Then, retrieve current data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to the Enterprise Platform with the session name `'myNS::remoteSession'` and the service name `'dIDN_RDF'` without DACS authentication. `c` is a Reuters connection object.

```
session = 'myNS::remoteSession';
servicename = 'dIDN_RDF';
c = reuters(session,servicename)
```

```
c =

  reuters with properties:

                          session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                             user: []
                         position: []
                      application: '182'
                       standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
                           client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
                      serviceName: 'dIDN_RDF'
                       eventQueue: [90 com.reuters.rfa.internal.common.EventQueueImpl]
              marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]
  marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]
                 mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                           defDb: [369 com.reuters.ts1.TS1DefDb]
```

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =

     PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains many Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`.

Close the Reuters connection.

```
close(c)
```

### Connect to Enterprise Platform Using DACS Authentication

Create an Enterprise Platform connection using the DACS authentication. Then, retrieve current data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to the Enterprise Platform using DACS authentication with the following:

- Session name `'myNS::remoteSession'`
- Service name `'dIDN_RDF'`
- User name `'ab123'`
- Data server IP address `'111.222.333.444/net'`

`c` is a Reuters connection object.

```
session = 'myNS::remoteSession';
servicename = 'dIDN_RDF';
```

```
user = 'ab123';
position = '111.222.333.444/net';

c = reuters(session,servicename, ...
    user,position)
```

```
c =

  reuters with properties:

                       session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                          user: 'mw335'
                      position: '111.222.333.444/net'
                   application: '182'
                    standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
                        client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
                   serviceName: 'dIDN_RDF'
                    eventQueue: [0 com.reuters.rfa.internal.common.EventQueueImpl]
           marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]
marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]
                mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                          defDb: []
```

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =

     PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains many Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`.

Close the Reuters connection.

```
close(c)
```

### Connect to Enterprise Platform for Real-Time Data Only

Create an Enterprise Platform connection for real-time data retrieval. Then, retrieve real-time data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to the Enterprise Platform with the session name `'myNS::remoteSession'` and the service name `'IDN_SELECTFEED'`. Leave the user name and DACS position blank. Specify the last argument as `1` to retrieve only real-time data. `c` is a Reuters connection object.

```
session = 'myNS::remoteSession';
servicename = 'IDN_SELECTFEED';

c = reuters(session,servicename,[],[],1)
```

```
c =

  reuters with properties:
```

```
                        session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                           user: []
                       position: []
                    application: '182'
                     standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
                         client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
                    serviceName: 'IDN_SELECTFEED'
                     eventQueue: [1 com.reuters.rfa.internal.common.EventQueueImpl]
            marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]
marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]
                mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                          defDb: []
```

Retrieve real-time data. The `rtdemo` event handler function returns the real-time Reuters data for the Google security to the MATLAB workspace variable `A`.

```
sec = 'GOOG.O';
eventhandler = 'rtdemo';

subs = fetch(c,sec,eventhandler);
```

Close the Reuters connection.

```
close(c)
```

### Connect to Enterprise Platform Using RTIC (TIC-RMDS Edition)

Create an Enterprise Platform connection using the RTIC (TIC-RMDS Edition). Then, retrieve current data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to the Enterprise Platform using an RTIC (TIC-RMDS Edition) connection without DACS authentication. Use the session name `'myNS::remoteRTICSession'` and the service name `'IDN_RDF'`. The Reuters connection object `c` appears in the MATLAB workspace.

```
session = 'myNS::remoteRTICSession';
servicename = 'IDN_RDF';

c = reuters(session,servicename)
```

```
c =

  reuters with properties:

                           session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                              user: []
                          position: []
                       application: '182'
                        standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
                            client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
                       serviceName: 'IDN_RDF'
                        eventQueue: [0 com.reuters.rfa.internal.common.EventQueueImpl]
               marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]
   marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]
                   mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                             defDb: []
```

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =
```

```
     PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains many Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`.

Close the Reuters connection.

```
close(c)
```

### Connect to Enterprise Platform Using RTIC (TIC-RMDS Edition) With DACS Authentication

Create an Enterprise Platform connection using the RTIC (TIC-RMDS Edition) with DACS authentication. Then, retrieve current data for a security. The current data you see when completing this example can differ from the output data shown.

Connect to the Enterprise Platform using an RTIC (TIC-RMDS Edition) connection with DACS authentication, and specify the following:

- Session name `'myNS::remoteRTICWithDACs'`
- Service name `'IDN_RDF'`
- User name `'ab123'`
- Data server IP address `'111.222.333.444/net'`

`c` is a Reuters connection object.

```
session = 'myNS::remoteRTICWithDACs';
servicename = 'IDN_RDF';
user = 'ab123';
position = '111.222.333.444/net';

c = reuters(session,servicename, ...
    user,position)
```

```
c =

  reuters with properties:

                          session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                             user: 'mw427'
                         position: '192.168.107.130'
                      application: '182'
                       standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
                           client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
                      serviceName: 'IDN_RDF'
                       eventQueue: [2 com.reuters.rfa.internal.common.EventQueueImpl]
             marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSubscriberImpl]
  marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInterestSpec]
                 mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                           defDb: []
```

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =

     PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains many Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`.

Close the Reuters connection.

```
close(c)
```

## Tips

- You can connect to the Reuters data server without DACS authentication. For example, use this code.

  ```
  c = reuters('myNS::remoteSession','IDN_CONFLATED');
  ```

- When you connect to the Enterprise Platform without DACS authentication, ignore these informational messages if they appear in the Command Window.

  ```
  Oct 5, 2007 2:28:31 PM
  com.reuters.rfa.internal.connection.
  ConnectionImpl initializeEntitlements
  INFO: com.reuters.rfa.connection.ssl....
              myNS.RemoteConnection
  DACS disabled for connection myNS::RemoteConnection
  ```

- When you connect to the Enterprise Platform with DACS authentication, ignore these informational messages if they appear in the Command Window.

  ```
  Oct 5, 2007 2:27:14 PM ...
  com.reuters.rfa.internal.connection.
  ConnectionImpl$ConnectionEstablishmentThread runImpl
  INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
  Connection successful: ...
                  componentName :myNS::RTICwithDacs,
  subscriberRVConnection:
  {service: 9453, network: 192.168.107.0;225.2.2.8,
  daemon: tcp:192.168.107.131:9450}
  Oct 5, 2007 2:27:14 PM
  com.reuters.rfa.internal.connection.sass3....
                  Sass3LoggerProxy log
  INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
  SASS3JNI: Received advisory from RV session@
  (9453,192.168.107.0;225.2.2.8,tcp:192.168.107.131:9450):
   _RV.INFO.SYSTEM.RVD.CONNECTED
  Oct 5, 2007 2:27:14 PM
  com.reuters.rfa.internal.connection.ConnectionImpl
  makeServiceInfo
  WARNING: com.reuters.rfa.connection.sass3....
                  myNS.RTICwithDacs
  Service list configuration has no
                  alias defined for network
  serviceName IDN_RDF
  ```

## See Also

`rmdsconfig`

**Topics**

“Configuring Enterprise Platform from Refinitiv Connections” on page 1-8
“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

**Introduced in R2008a**

# addric

Create Reuters Instrument Code

## Syntax

```
addric(c,ric,fid,fval,type)
```

## Description

`addric(c,ric,fid,fval,type)` creates a Reuters Instrument Code, `ric`, on the service defined by the Reuters session, `c`. Supply the field ID or name, `fid`, and the field value, `fval`. Specify whether the RIC `type` is `'live'` or `'static'` (default).

## Examples

Create a live RIC called `myric` with the fields `'trdprc_1'` (field ID 6) and `'bid'` (field ID 22) set to initial values of `0`:

```
addric(c,'myric',{trdprc_1','bid'},{0,0},'live')
```

Create a live RIC called `myric` with the fields `trdprc_1` and `bid` set to initial values of `0`:

```
addric(c,'myric',{6,22},{0,0},'live')
```

## See Also

`contrib` | `deleteric` | `fetch` | `reuters`

**Topics**

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

**Introduced in R2010b**

# close

Release connections to Reuters data servers

## Syntax

```
close(c)
```

## Arguments

| | |
|---|---|
| `c` | Reuters session object created with `reuters`. |

## Description

`close(c)` releases the Enterprise Platform from Refinitiv connection `c`.

## Examples

Release the connection `c` to the Reuters data server, and unsubscribe all requests associated with it:

```
close(c)
```

## See Also

`reuters`

### Topics

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

### Introduced in R2008a

# contrib

Contribute data to Reuters data feed

## Syntax

```
contrib(c,s,fid,fval)
```

## Description

`contrib(c,s,fid,fval)` contributes data to a Reuters data feed. `c` is the Reuters session object, and `s` is the RIC. Supply the field IDs or names, `fid`, and field values, `fval`.

## Examples

Contribute data to the Reuters datafeed for the Reuters session object `c` and the RIC `'myric'`. Provide a last trade price of 33.5.

```
contrib(c,'myric','trdprc_1',33.5)
```

Contribute an additional bid price of 33.8:

```
contrib(c,'myric',{'trdprc_1','bid'},{33.5,33.8})
```

Submit value 33.5 for field 6 (`'trdprc_1'`):

```
contrib(c,'myric',6,33.5)
```

Add the value 33.8 to field 22 (`'bid'`):

```
contrib(c,'myric',{6,22},{33.5,33.8})
```

## See Also

`addric` | `deleteric` | `fetch` | `reuters`

**Topics**

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

**Introduced in R2010b**

# deleteric

Delete Reuters Instrument Code

## Syntax

```
deleteric(c,ric)
deleteric(c,ric,fid)
```

## Description

`deleteric(c,ric)` deletes the Reuters Instrument Code, `ric`, and all associated fields. `c` is the Reuters session object.

`deleteric(c,ric,fid)` deletes the fields specified by `fid` for the `ric`.

## Examples

Delete `'myric'` and all of its fields:

```
deleteric(c,'myric')
```

Delete the fields `'fid1'` and `'fid2'` from `'myric'`:

```
deleteric(c,'myric',{'fid1','fid2'})
```

## See Also

`addric` | `contrib` | `fetch` | `reuters`

### Topics

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

### Introduced in R2010b

# fetch

Request data from Reuters data servers

## Syntax

```
d = fetch(c,sec)
d = fetch(c,sec,[],fields)
subs = fetch(c,sec,eventhandler)
```

## Description

`d = fetch(c,sec)` returns the current data for the security `sec`, given the Reuters session object `c`.

`d = fetch(c,sec,[],fields)` requests the given fields `fields`, for the security `sec`, given the Reuters session object `c`.

`subs = fetch(c,sec,eventhandler)` uses the Reuters session object `c` to subscribe to the security `sec`. MATLAB runs the `eventhandler` function for each data event that occurs.

## Examples

### Retrieve Current Securities Data

Connect to Refinitiv.

```
c = reuters('myNS::remotesession','dIDN_RDF');
```

```
Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements

INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output message specifies a successful connection to the Enterprise Platform.

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)
```

```
d =

     PROD_PERM: 74.00
    RDNDISPLAY: 66.00
    DSPLY_NAME: 'DELAYED-15GOOGLE'
    ...
```

`d` contains a large number of Refinitiv market data fields. This output shows the product permissions information, `PROD_PERM`, the display information for the IDN terminal device, `RDNDISPLAY`, and the expanded name for the instrument, `DSPLY_NAME`.

Close the Refinitiv connection.

```
close(c)
```

### Request Specific Fields

Connect to Refinitiv.

```
c = reuters('myNS::remotesession','dIDN_RDF');
```

```
Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements

INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output specifies a successful connection to the Enterprise Platform.

Request the product permissions information `'PROD_PERM'` for the Google security from Reuters.

```
sec = 'GOOG.O';
field = 'PROD_PERM';

d = fetch(c,sec,[],field)
```

```
d =

    PROD_PERM: 74
```

Request the product permissions information `'PROD_PERM'` and the display information for the IDN terminal device `'RDNDISPLAY'` for the Google security from Reuters. Use a cell array to input these two fields to the function.

```
sec = 'GOOG.O';
fields = {'PROD_PERM','RDNDISPLAY'};

d = fetch(c,sec,[],fields)
```

```
d =

     PROD_PERM: 74
    RDNDISPLAY: 66
```

Close the Refinitiv connection.

```
close(c)
```

### Subscribe to a Security

To subscribe to a security and process the data in real time, specify an event handler function. MATLAB runs this function each time it receives a real-time data event from Reuters.

Connect to Refinitiv.

```
c = reuters('myNS::remotesession','dIDN_RDF');
```

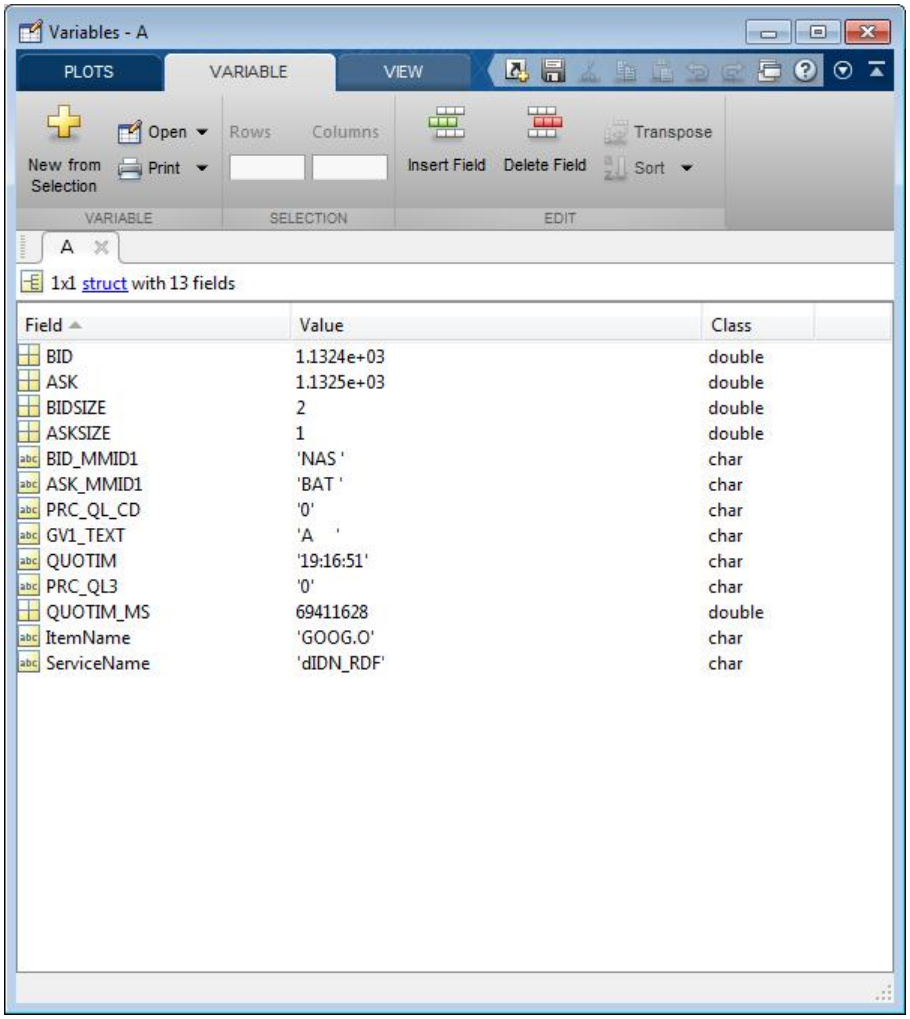
```
Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements
```

```
INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output specifies a successful connection to the Enterprise Platform.

The event handler `rtdemo` function returns the real-time Reuters data for the Google security to the MATLAB workspace variable `A`. `openvar` displays `A` in the Variables editor.

```
sec = 'GOOG.O';
eventhandler = 'rtdemo';

subs = fetch(c,sec,eventhandler);
openvar('A')
```




In this instance, the fields represent a bid or ask tick.

The `fetch` function returns the subscription handle associated with this request in the variable `subs`. Display the subscription handle contents.

```
subs
```

```
subs =
```

```
com.reuters.rfa.internal.common.SubHandleImpl[]:
    [com.reuters.rfa.internal.common.SubHandleImpl]
```

Stop the real-time subscription.

```
stop(c,subs)
```

Close the Refinitiv connection.

```
close(c)
```

## Input Arguments

### `c` — Reuters session

object

Reuters session, specified as a Reuters session object created using `reuters`.

### `sec` — Security list

character vector | string scalar | cell array of character vectors | string array

Security list, specified as a character vector, string scalar, cell array of character vectors, or a string array to denote Reuters securities.

Data Types: `char` | `cell` | `string`

### `fields` — Reuters fields list

character vector | string scalar | cell array of character vectors | string array

Reuters fields list, specified as a character vector, string scalar, cell array of character vectors, or string array to denote Reuters field names.

Data Types: `char` | `cell` | `string`

### `eventhandler` — Reuters real-time event handler

function

Reuters real-time event handler, specified as a MATLAB function that runs for each data event that occurs. The sample event handler called `rtdemo.m` returns Reuters real-time data from the Enterprise Platform from Refinitiv to the MATLAB workspace. The sample event handler specifies these input arguments.

| Event Handler Input Argument | Description |
|---|---|
| `x` | Return data structure |
| `itemName` | Reuters data name |
| `serviceName` | Reuters service name |

The sample event handler writes variable `A` to the Workspace browser with the contents of `x`.

Data Types: `function_handle`

## Output Arguments

### `d` — Reuters request data

structure

Reuters request data, returned as a structure. The structure contains many Reuters data fields. For details, see *Reuters Data Support*.

**subs — Reuters subscription handle**
object

Reuters subscription handle, returned as a Reuters subscription object.

## See Also

`close` | `reuters` | `stop`

**Topics**

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17
“Writing and Running Custom Event Handler Functions” on page 1-26

**Introduced in R2008a**

# get

Retrieve properties of Reuters session objects

## Syntax

```
e = get(c)
e = get(c,f)
```

## Arguments

| | |
|---|---|
| `c` | Reuters session object created with `reuters`. |
| `f` | Reuters session properties list. |

## Description

`e = get(c)` returns Reuters session properties for the Reuters session object `c`.

`e = get(c,f)` returns Reuters session properties specified by the properties list `f` for the Reuters session object `c`.

## See Also

`reuters`

### Topics

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

### Introduced in R2008a

# history

Request data from Reuters Time Series One

## Syntax

```
d = history(c,s)
d = history(c,s,p)
d = history(c,s,f)
d = history(c,s,f,p)
d = history(c,s,d)
d = history(c,s,startdate,enddate)
d = history(c,s,startdate,enddate,p)
d = history(c,s,f,startdate,enddate)
d = history(c,s,f,startdate,enddate,p)
```

## Description

`d = history(c,s)` returns all available daily historical data for the RIC, `s`, for the Reuters session object `c`.

`d = history(c,s,p)` returns all available historical data for the RIC, `s`, for the Reuters session object `c`. `p` specifies the period of the data:

- `'d'` - daily (default)
- `'w'` - weekly
- `'m'` - monthly

**Note** Reuters Time Series One will only return two years of daily data, five years of weekly data, or ten years of monthly data from the current date.

`d = history(c,s,f)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`.

`d = history(c,s,f,p)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`. `p` specifies the period of the data.

`d = history(c,s,d)` returns the historical data for the RIC, `s`, for the given date, `d`, for the Reuters session object `c`.

`d = history(c,s,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,startdate,enddate,p)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

`d = history(c,s,f,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,f,startdate,enddate,p)` returns the historical data for the RIC, `s`, and fields, `f`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

## Examples

`d = history(c,'WXYZ.O')` returns a structure containing all available historical end of day daily data for the RIC `'WXYZ.O'`, for the Reuters session object `c`.

`d = history(c,'WXYZ.O','close')` returns a structure with the fields `date` and `close` containing all available historical end of day daily data for the RIC `'WXYZ.O'`.

`d = history(c,'WXYZ.O','close','m')` returns all available monthly data.

`d = history(c,'WXYZ.O','01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009. Note that only two years worth of daily data, five years worth of weekly data, and 10 years of monthly data from today's date is made available by Reuters.

`d = history(c,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

`d = history(c,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009','w')` returns all available weekly data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

## See Also

`close` | `fetch` | `reuters`

**Topics**

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

**Introduced in R2009b**

# stop

Unsubscribe securities

## Syntax

```
stop(c)
stop(c,d)
```

## Arguments

| | |
|---|---|
| c | Reuters session object created with `reuters`. |
| d | Subscription handle returned by `fetch`. |

## Description

`stop(c)` unsubscribes all securities associated with the Reuters session object `c`.

`stop(c,d)` unsubscribes the securities associated with the subscription handle `d`, where `d` is the subscription handle returned by `reuters/fetch`.

## Examples

Unsubscribe securities associated with a specific request `d` and a Reuters connection object `c`:

```
stop(c,d)
```

Unsubscribe all securities associated with the Reuters connection object `c`:

```
stop(c)
```

## See Also

`fetch` | `reuters`

**Topics**

“Retrieve Current and Historical Data Using Enterprise Platform from Refinitiv” on page 1-17

**Introduced in R2008a**

# rmdsconfig

Enterprise Platform from Refinitiv configuration editor

## Syntax

```
rmdsconfig
```

## Description

`rmdsconfig` opens the Enterprise Platform from Refinitiv configuration editor.

## See Also

`reuters`

**Introduced in R2010b**

# rnseloader

Retrieve data from Machine Readable News from Refinitiv sentiment archive file

## Syntax

```
x = rnseloader(file)
x = rnseloader(file, 'date', {DATE1})
x = rnseloader(file, 'date', {DATE1, DATE2})
x = rnseloader(file, 'security', {SECNAME})
x = rnseloader(file, 'start', STARTREC)
x = rnseloader(file, 'records', NUMRECORDS)
x = rnseloader(file, 'fieldnames', F)
```

## Arguments

Specify the following arguments as name-value pairs. You can specify any combination of name-value pairs in a single call to `rnseloader`.

| | |
|---|---|
| `file` | Machine Readable News sentiment archive file from which to retrieve data. |
| `'date'` | Use this argument with `{DATE1,DATE2}` to retrieve data between and including the specified dates. Specify the dates as numbers, character vectors, or strings. |
| `'security'` | Use this argument to retrieve data for `SECNAME`, where `SECNAME` is a cell array containing a list of security identifiers for which to retrieve data. |
| `'start'` | Use this argument to retrieve data beginning with the record `STARTREC`, where `STARTREC` is the record at which `rnseloader` begins to retrieve data. Specify `STARTREC` as a number. |
| `'records'` | Use this argument to retrieve `NUMRECORDS` number of records. |

## Description

`x = rnseloader(file)` retrieves data from the Machine Readable News sentiment archive file `file`, and stores it in the structure `x`.

`x = rnseloader(file, 'date', {DATE1})` retrieves data from `file` with date stamps of value `DATE1`.

`x = rnseloader(file, 'date', {DATE1, DATE2})` retrieves data from `file` with date stamps between `DATE1` and `DATE2`.

`x = rnseloader(file, 'security', {SECNAME})` retrieves data from `file` for the securities specified by `SECNAME`.

`x = rnseloader(file, 'start', STARTREC)` retrieves data from `file` beginning with the record specified by `STARTREC`.

`x = rnseloader(file, 'records', NUMRECORDS)` retrieves `NUMRECORDS` number of records from `file`.

`x = rnseloader(file, 'fieldnames', F)` retrieves only the specified fields, `F`, in the output structure.

## Examples

Retrieve data from the file `'file.csv'` with date stamps of `'02/02/2007'`:

```
x = rnseloader('file.csv','date',{'02/02/2007'})
```

Retrieve data from `'file.csv'` between and including `'02/02/2007'` and `'02/03/2007'`:

```
 x = rnseloader('file.csv','date',{'02/02/2007',...
'02/03/2007'})
```

Retrieve data from `'file.csv'` for the security `'XYZ.O'`:

```
 x = rnseloader('file.csv','security',{'XYZ.O'})
```

Retrieve the first 10000 records from `'file.csv'`:

```
 x = rnseloader('file.csv','records',10000)
```

Retrieve data from `'file.csv'`, starting at record 100000:

```
  x = rnseloader('file.csv','start',100000)
```

Retrieve up to 100000 records from `'file.csv'`, for the securities `'ABC.N'` and `'XYZ.O'`, with date stamps between and including the dates `'02/02/2007'` and `'02/03/2007'`:

```
x = rnseloader('file.csv','records',100000,...
               'date',{'02/02/2007','02/03/2007'},...
               'security',{'ABC.N','XYZ.O'})
```

## See Also

`rdthloader` | `reuters`

**Introduced in R2008b**

# tlkrs

SIX Financial Information connection

# Description

The `tlkrs` function creates a `tlkrs` object. The `tlkrs` object represents a SIX Financial Information connection.

After you create a `tlkrs` object, you can use the object functions to retrieve current and intraday tick data.

# Creation

## Syntax

```
c = tlkrs(ci,ui,password)
```

### Description

`c = tlkrs(ci,ui,password)` creates a connection to the SIX Financial Information data service and sets the ci, ui and password properties.

## Properties

### `ci` — Customer identifier

character vector | string scalar

Customer identifier, specified as a character vector or string scalar. For credentials, contact SIX Financial Information.

Example: `'US12345'`

Data Types: `char` | `string`

### `ui` — User identifier

character vector | string scalar

User identifier, specified as a character vector or string scalar. For credentials, contact SIX Financial Information.

Example: `'userapid01'`

Data Types: `char` | `string`

### `password` — Password

character vector | string scalar

Password, specified as a character vector or string scalar. For credentials, contact SIX Financial Information.

Example: `'userapid10'`

Data Types: `char` | `string`

**`sessionid` — Session identifier**
character vector

This property is read-only.

Session identifier, specified as a character vector. The `tlkrs` function determines the session identifier from the file specified by the login URL.

Example: `'463487494'`

Data Types: `char`

## Object Functions

| | |
|---|---|
| close | Close connection to SIX Financial Information |
| getdata | Current SIX Financial Information data |
| history | End of day SIX Financial Information data |
| timeseries | SIX Financial Information intraday tick data |
| isconnection | Determine if SIX Financial Information connection is valid |
| tkfieldtoid | SIX Financial Information field names to identification string |
| tkidtofield | SIX Financial Information identification string to field name |

## Examples

### Connect to SIX Financial Information

Create a SIX Financial Information connection. Then, retrieve current data for instruments. The current data you see when completing this example can differ from the output data shown.

Create a SIX Financial Information connection with a customer identifier, user identifier, and password. `c` is a `tlkrs` object.

```
ci = 'US12345';
ui = 'userapid01';
password = 'userapid10';
c = tlkrs(ci,ui,password)
```

```
c =

  tlkrs with properties:

          ci: 'US12345'
          ui: 'userapid01'
    password: 'userapid10'
   sessionid: '463487494'
```

Convert the bid, ask, and last price fields to the identifiers `ids`.

```
f = {'Bid','Ask','Last'};
typ = 'market';
ids =  tkfieldtoid(c,f,typ);
```

Retrieve the current data for the specified securities `s`.

```
s = {'1758999,149,134','275027,148,184'};
d = getdata(c,s,ids);
```

Display the pricing data.

```
d.P.v
```

```
ans =

  6×1 cell array

    {'64.36'  }
    {'64.37'  }
    {0×0 double}
    {'1.26' }
    {'1.26' }
    {0×0 double}
```

Close the SIX Financial Information connection.

```
close(c)
```

## See Also

**Introduced in R2011b**

# close

Close connection to SIX Financial Information

## Syntax

```
close(C)
```

## Description

`close(C)` closes the connection, `C`, to SIX Financial Information.

## See Also

`tlkrs`

**Introduced in R2011b**

# getdata

Current SIX Financial Information data

## Syntax

```
D = getdata(c,s,f)
```

## Description

`D = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`.

## Examples

Retrieve SIX Financial Information pricing data for specified securities.

```
% Connect to Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert specified fields to ID strings.
ids =  tkfieldtoid(c,{'Bid','Ask','Last'},'market');

% Retrieve data for specified securities.
d = getdata(c,{'1758999,149,134','275027,148,184'},ids);
```

Your output appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, and `d.P` contains the pricing data.

View the instrument IDs like this:

```
d.I.k

ans =
    '1758999,149,134'
    '275027,148,184'
```

View the pricing data field IDs like this:

```
d.P.k

ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
```

```
    '33,3,1'
    '3,1,1'
```

And the pricing data like this:

```
d.P.v

ans =

    '44.94'
    '44.95'
    []
    '0.9715'
    '0.9717'
    []
```

Convert field IDs in `d.P.k` to field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'market')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names (`Bid`, `Ask`, `Last`) and corresponding IDs.

## See Also

`history` | `timeseries` | `tkfieldtoid` | `tkidtofield` | `tlkrs`

**Introduced in R2011b**

# history

End of day SIX Financial Information data

## Syntax

```
D = history(c,s,f,fromdate,todate)
```

## Description

`D = history(c,s,f,fromdate,todate)` returns the historical data for the security list `s`, for the fields `f`, for the dates `fromdate` to `todate`.

## Examples

Retrieve end of day SIX Financial Information data for the specified security for the past 5 days.

```
c = tlkrs('US12345','userapid01','userapid10')
ids = tkfieldtoid(c,{'Bid','Ask'},'history');
d = history(c,{'1758999,149,134'},ids,floor(now)-5,floor(now));

d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
     HL: [1x1 struct]
     HD: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.HD` contains the dates, and `d.P` contains the pricing data.

View the dates:

```
d.HD.d

ans =

    '20110225'
    '20110228'
    '20110301'
```

View the pricing field IDs:

```
d.P.k

ans =

    '3,2'
    '3,3'
    '3,2'
    '3,3'
    '3,2'
    '3,3'
```

View the pricing data:

```
d.P.v

ans =

    '45.32'
    '45.33'
    '45.26'
    '45.27'
    '44.94'
    '44.95'
```

Convert the field identification values in `d.P.k` to their corresponding field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

## See Also

`getdata` | `timeseries` | `tkfieldtoid` | `tkidtofield` | `tlkrs`

**Introduced in R2011b**

# isconnection

Determine if SIX Financial Information connection is valid

## Syntax

```
X = isconnection(C)
```

## Description

`X = isconnection(C)` returns `true` if `C` is a valid SIX Financial Information connection and `false` otherwise.

## See Also

`close` | `getdata` | `tlkrs`

**Introduced in R2011b**

# timeseries

SIX Financial Information intraday tick data

## Syntax

```
D = timeseries(c,s,t)
D = timeseries(c,s,{startdate,enddate})
D = timeseries(c,s,t,5)
```

## Description

`D = timeseries(c,s,t)` returns the raw tick data for the SIX Financial Information connection object `c`, the security `s`, and the date `t`. Every trade, best, and ask tick is returned for the given date or date range.

`D = timeseries(c,s,{startdate,enddate})` returns the raw tick data for the security `s`, for the date range defined by `startdate` and `enddate`.

`D = timeseries(c,s,t,5)` returns the tick data for the security `s`, for the date `t` in intervals of 5 minutes, for the field `f`. Intraday tick data requested is returned in 5-minute intervals, with the columns representing:

- First
- High
- Low
- Last
- Volume weighted average
- Moving average

## Examples

Retrieve SIX Financial Information intraday tick data for the past 2 days:

```
c = tlkrs('US12345','userapid01','userapid10')
d = timeseries(c,{'1758999,149,134'}, ...
   {floor(now)-.25,floor(now)})
```

Display the returned data:

```
d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
    TSL: [1x1 struct]
     TS: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.TS` contains the date and time data, and `d.P` contains the pricing data.

Display the tick times:

```
d.TS.t(1:10)

ans =

    '013500'
    '013505'
    '013510'
    '013520'
    '013530'
    '013540'
    '013550'
    '013600'
    '013610'
    '013620'
```

Display the field IDs:

```
d.P.k(1:10)

ans =

    '3,4'
    '3,2'
    '3,3'
    '3,4'
    '3,2'
    '3,3'
    '3,4'
    '3,2'
    '3,3'
    '3,4'
```

Convert these IDs to field names (`Mid`, `Bid`, `Ask`) with `tkidtofield`:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names and corresponding IDs.

Display the corresponding tick values:

```
d.P.v(1:10)

ans =

    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
```

## See Also

`getdata` | `history` | `tlkrs`

**Introduced in R2011b**

# tkfieldtoid

SIX Financial Information field names to identification string

## Syntax

```
D = tkfieldtoid(c,f,typ)
```

## Description

`D = tkfieldtoid(c,f,typ)` converts SIX Financial Information field names to their corresponding identification strings. `c` is the SIX Financial Information connection object, `f` is the field list, and `typ` denotes the field. Options for the field include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with `history`.

## Examples

Retrieve pricing data associated with specified identification strings:

```
% Connect to SIX Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert field names to identification strings.
ids = tkfieldtoid(c,{'bid','ask','last'},'market');

% Retrieve data associated with the identification strings.
d = getdata(c,{'1758999,149,134','275027,148,184',ids);
```

## See Also

`getdata` | `history` | `timeseries` | `tkidtofield` | `tlkrs`

**Introduced in R2011b**

# tkidtofield

SIX Financial Information identification string to field name

## Syntax

```
D = tkidtofield(c,f,typ)
```

## Description

`D = tkidtofield(c,f,typ)` converts SIX Financial Information field identification strings to their corresponding field names. `c` is the SIX Financial Information connection object, `f` is the ID list, and `typ` denotes the fields. Options for the fields include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with the `history`.

## Examples

When you retrieve output from SIX Financial Information, it appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

The instrument IDs are found in `d.I`, and the pricing data is found in `d.P`. The output for `d.P.k` appears like this:

```
ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
    '33,3,1'
    '3,1,1'
```

Convert the field IDs in `d.P.k` to their field names with the `tkidtofield` function:

```
d.P.k = tkidtofield(c,d.P.k,'market')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names and their corresponding field IDs.

## See Also

`getdata` | `history` | `timeseries` | `tkfieldtoid` | `tlkrs`

**Introduced in R2011b**

# statsllc

STATS.com connection

# Description

The `statsllc` function creates a `statsllc` object. The `statsllc` object represents a STATS.com connection.

After you create a `statsllc` object, you can use the object functions to retrieve athlete, team, and event data. You can retrieve data for the sport and time period that you have access to using your credentials. The credentials consist of an API key and secret pass code. To retrieve data for different athletes, teams, events, and time periods, specify query parameters. For credentials, contact STATS.com. For all available query parameters, see STATS Developer Center I/O Docs.

# Creation

## Syntax

```
c = statsllc(apikey,secret)
c = statsllc(apikey,secret,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

### Description

`c = statsllc(apikey,secret)` creates a STATS.com connection object using an API key and secret pass code. This syntax does not set the properties of the connection object. To retrieve data with a custom URL, use this syntax with `fetchUrl`. To retrieve data without a custom URL, use this syntax, set the object properties, and run `fetch`.

`c = statsllc(apikey,secret,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` creates a STATS.com connection using web service query parameters, as specified by one or more name-value pair arguments. The STATS.com web service defines the query parameters. For details, see STATS Developer Center I/O Docs. To retrieve data, use this syntax with `fetch`.

### Input Arguments

**`apikey` — STATS.com API key**
character vector | string scalar

STATS.com API key, specified as a character vector or string scalar. To request your API key, contact STATS.com.

Data Types: `char` | `string`

**`secret` — STATS.com secret pass code**
character vector | string scalar

STATS.com secret pass code, specified as a character vector or string scalar. To request your secret pass code, contact STATS.com.

Data Types: `char` | `string`

**`QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters**
name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of the query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

The web service defines name-value pairs that it accepts as part of a request. For valid name-value pairs, see STATS Developer Center I/O Docs. To understand which name-value pairs you have access to, check your license or contact STATS.com.

Example: `'DataType','stats','LeagueAbbreviation','mlb','Resource','standings','SportName','baseball','VersionNumber','v1'` retrieves the standings for the current or most recent baseball season.

Data Types: `char` | `string`

## Properties

**`DataType` — STATS.com type of data**
character vector

STATS.com type of data, specified as a character vector.

The type `'stats'` means statistical data. The type `'optical'` means spatial data derived from video recordings. For other types of data, contact STATS.com.

Data Types: `char`

**`LeagueAbbreviation` — Abbreviation for sports league name**
character vector

Abbreviation for a sports league name, specified as a character vector.

Example: `'mlb'`

Data Types: `char`

**`Method` — Player or team identifier**
character vector

Player or team identifier, specified as a character vector.

Example: `'548033'`

Data Types: `char`

**`Resource` — Type of information to request**
character vector

Type of information to request, specified as a character vector.

To request data for games, specify `'events'`. To request statistical data for a player, specify `'stats/players'`. For other resources, contact STATS.com.

Data Types: `char`

**`SportName` — Sport name**
character vector

Sport name, specified as a character vector.

Example: `'baseball'`

Data Types: `char`

**`URL` — URL**
`'http://api.stats.com'` (default) | character vector

URL, specified as a character vector. STATS.com uses a custom-built URL to make the data request. For details about the URL, contact STATS.com.

The base URL stub for creating the web request is `'http://api.stats.com'`.

Data Types: `char`

**`VersionNumber` — STATS.com version number**
character vector

STATS.com version number, specified as a character vector. For details about the STATS.com version number, contact STATS.com.

Example: `'v1'`

Data Types: `char`

## Object Functions

| | |
|---|---|
| fetch | Retrieve data from STATS.com |
| fetchUrl | Retrieve data from STATS.com with URL |

## Examples

### Connect to STATS.com

Create a STATS.com connection by using credentials, and then retrieve basketball statistics for individual players.

Connect to STATS.com using an API key and secret pass code. `c` is the STATS.com connection object.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

c = statsllc(apikey,secret)
```

```
c =

  statsllc with properties:

             DataType: []
    LeagueAbbreviation: []
               Method: []
```

```
              Resource: []
             SportName: []
                   URL: 'http://api.stats.com'
         VersionNumber: []
```

Retrieve basketball statistics for individual players in the sport league named `'nba'`. To specify the statistics for retrieval, create a URL suffix.

```
urlsuffix = '/v1/stats/basketball/nba/participants';

d = fetchUrl(c,urlsuffix);
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetchUrl`.

Retrieving data from STATS.com indicates a successful connection.

### Connect to STATS.com by Setting Object Properties

Create a STATS.com connection by setting object properties, and then retrieve basketball team data.

Connect to STATS.com using an API key and secret pass code. `c` is the STATS.com connection object.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

c = statsllc(apikey,secret)
```

```
c =

  statsllc with properties:

              DataType: []
    LeagueAbbreviation: []
                Method: []
              Resource: []
             SportName: []
                   URL: 'http://api.stats.com'
         VersionNumber: []
```

Set the object properties in `sBasketball` to create a specific data request. Specify statistical standings data for basketball teams in the sport league named `'nba'`.

```
c.DataType = 'stats';
c.LeagueAbbreviation = 'nba';
c.Resource = 'standings';
c.SportName = 'basketball';
c.VersionNumber = 'v1';
```

Retrieve basketball team data using the STATS.com connection `c`.

```
d = fetch(c);
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetch`.

Retrieving data from STATS.com indicates a successful connection.

### Connect to STATS.com with Query Parameters

Create a STATS.com connection using query parameters, and then retrieve basketball team data for a season.

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters. `c` is the STATS.com connection object.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

c = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1')
```

```
c =

  statsllc with properties:

             DataType: 'stats'
    LeagueAbbreviation: 'nba'
               Method: []
             Resource: 'standings'
            SportName: 'basketball'
                  URL: 'http://api.stats.com'
        VersionNumber: 'v1'
```

Retrieve the basketball team data for the 2015 season.

```
d = fetch(c,'season','2015');
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetch`.

Retrieving data from STATS.com indicates a successful connection.

## See Also

### Topics

“Compare Player Salaries by Injury Status” on page 3-25
“Retrieve Team Standings for the Current Year” on page 3-28
“STATS.com Data Retrieval Errors” on page 4-2

### External Websites

STATS.com
STATS Developer Center I/O Docs

### Introduced in R2016b

# fetch

Retrieve data from STATS.com

## Syntax

```
d = fetch(c)
d = fetch(c,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

## Description

`d = fetch(c)` returns STATS.com data using the STATS.com connection `c`.

`d = fetch(c,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` returns STATS.com data using web service query parameters, as specified by one or more name-value pair arguments. The STATS.com web service defines the query parameters. For query parameters, see STATS Developer Center I/O Docs.

## Examples

### Retrieve STATS.com Data

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1');
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball team data using the STATS.com connection `sBasketball`.

```
d = fetch(sBasketball)
```

```
d =

  struct with fields:

            status: 'OK'
       recordCount: 30
    startTimestamp: '2016-04-19T18:37:36.0745302Z'
      endTimestamp: '2016-04-19T18:37:36.9026552Z'
         timeTaken: 0.8281
        apiResults: [1×1 struct]
```

The query returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request
- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve the basketball team data `f` for the second conference in the structure array.

```
f = d.apiResults.league.season.eventType.conferences(2)
```

```
f =

  struct with fields:

    conferenceId: 2
            name: 'Western Conference'
    abbreviation: 'Western'
       divisions: [3×1 struct]
```

`f` is a structure with these fields:

- `conferenceId` — Conference identifier
- `name` — Conference name
- `abbreviation` — Conference abbreviation
- `divisions` — Divisions in the conference

Retrieve the team identifiers in the first division in the structure array `divisions`.

```
f.divisions(1).teams.teamId
```

```
ans =

     9

ans =

    12
...
```

### Retrieve STATS.com Data Using Query Parameters

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1');
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball team data using the STATS.com connection `sBasketball` for the 2015 season.

```
d = fetch(sBasketball,'season','2015')
```

```
d =

  struct with fields:

           status: 'OK'
      recordCount: 30
    startTimestamp: '2016-04-19T18:37:36.0745302Z'
      endTimestamp: '2016-04-19T18:37:36.9026552Z'
         timeTaken: 0.8281
        apiResults: [1×1 struct]
```

The query returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request
- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve the basketball team data `f` for the second conference in the structure array.

```
f = d.apiResults.league.season.eventType.conferences(2)
```

```
f =

  struct with fields:

    conferenceId: 2
            name: 'Western Conference'
    abbreviation: 'Western'
       divisions: [3×1 struct]
```

`f` is a structure with these fields:

- `conferenceId` — Conference identifier
- `name` — Conference name
- `abbreviation` — Conference abbreviation
- `divisions` — Divisions in the conference

Retrieve the team identifiers in the first division in the structure array `divisions`.

```
f.divisions(1).teams.teamId
```

```
ans =

     9

ans =
```

```
    12
...
```

## Input Arguments

### c — STATS.com connection

`statsllc` object

STATS.com connection, specified as a `statsllc` object.

### QueryName1,QueryValue1,...,QueryNameN,QueryValueN — Web service query parameters

name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of a query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

The web service defines name-value pairs that it accepts as part of a request. For valid name-value pairs, see STATS Developer Center I/O Docs. To understand which name-value pairs you have access to, contact STATS.com.

The name-value pairs for this function are different from the name-value pairs specified in `statsllc`. For details about the differences, contact STATS.com.

Example: `'season','2015'` retrieves data for the 2015 season.

Data Types: `char` | `string`

## Output Arguments

### d — STATS.com data

structure

STATS.com data, returned as a structure with these fields:

| STATS.com Data Structure Field | Description |
|---|---|
| `status` | Web request status (`'OK'` denotes a successful web request) |
| `recordCount` | Number of records returned |
| `startTimestamp` | Start date and time of the web request |
| `endTimestamp` | End date and time of the web request |
| `timeTaken` | Number of seconds to complete the web request |
| `apiResults` | Returned data |

To retrieve individual or team athletic data, access the field `apiResults` in the structure. For example, enter:

```
d.apiResults
```

```
ans =
```

```
  struct with fields:

    sportId: 1
       name: 'Basketball'
     league: [1×1 struct]
```

For more data, continue to access the structure fields that contain further data. For accessing nested structures, see “Access Data in Nested Structures”.

## See Also

`fetchUrl` | `statsllc`

**Topics**

“Compare Player Salaries by Injury Status” on page 3-25
“Retrieve Team Standings for the Current Year” on page 3-28
“STATS.com Data Retrieval Errors” on page 4-2

**External Websites**

STATS.com
STATS Developer Center I/O Docs

**Introduced in R2016b**

# fetchUrl

Retrieve data from STATS.com with URL

## Syntax

```
d = fetchUrl(c,urlsuffix)
```

## Description

`d = fetchUrl(c,urlsuffix)` returns STATS.com data using the STATS.com connection `c` and a custom URL suffix.

## Examples

### Retrieve STATS.com Data Using Custom URL

Connect to STATS.com using an API key and secret pass code.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret);
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball statistics for individual players in the sport league named `'nba'`. To specify the statistics for retrieval, create a URL suffix.

```
urlsuffix = '/v1/stats/basketball/nba/participants';

d = fetchUrl(sBasketball,urlsuffix)
```

```
d =

  struct with fields:

             status: 'OK'
        recordCount: 450
     startTimestamp: '2016-04-26T15:28:46.2203857Z'
       endTimestamp: '2016-04-26T15:28:46.892265Z'
          timeTaken: 0.6719
         apiResults: [1×1 struct]
```

STATS.com returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request

- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve data about the first player in the league from the returned data `d.apiResults`. Access the nested structure to retrieve data.

```
d.apiResults.league.players{1}
```

```
ans =

  struct with fields:

    isSuspended: 0
      isInjured: 1
     highSchool: [1×1 struct]
...
```

The structure result contains suspended status, injury status, and high school information along with other fields. For details about these fields, contact STATS.com.

For more data, continue to access the nested structures or structure arrays. For accessing nested structures, see “Access Data in Nested Structures”.

## Input Arguments

### `c` — STATS.com connection

`statsllc` object

STATS.com connection, specified as a `statsllc` object.

### `urlsuffix` — URL suffix

character vector | string scalar

URL suffix, specified as a character vector or string scalar. `fetchUrl` builds a custom URL by appending the URL suffix to the connection URL created using `statsllc`. The `fetchUrl` function uses the custom URL to retrieve STATS.com data. To compose the URL suffix, specify query parameter values. For query parameter values, see STATS Developer Center I/O Docs.

Example: `'/v1/stats/baseball/mlb/participants'` retrieves statistical data for individual baseball players in the sports league named `'mlb'`

Data Types: `char` | `string`

## Output Arguments

### `d` — STATS.com data

structure

STATS.com data, returned as a structure with these fields:

| STATS.com Data Structure Field | Description |
|---|---|
| `status` | Web request status (`'OK'` denotes a successful web request) |

| STATS.com Data Structure Field | Description |
|---|---|
| `recordCount` | Number of records returned |
| `startTimestamp` | Start date and time of the web request |
| `endTimestamp` | End date and time of the web request |
| `timeTaken` | Number of seconds to complete the web request |
| `apiResults` | Returned data |

To retrieve individual or team athletic data, access the field `apiResults` in the structure. For example, enter:

```
d.apiResults
```

```
ans =

  struct with fields:

    sportId: 1
       name: 'Basketball'
     league: [1×1 struct]
```

For more data, continue to access the structure fields that contain further data. For accessing nested structures, see “Access Data in Nested Structures”.

## See Also

`fetch` | `statsllc`

**Topics**

“STATS.com Data Retrieval Errors” on page 4-2

**External Websites**

STATS.com
STATS Developer Center I/O Docs

**Introduced in R2016b**

# twitter

Twitter connection object

## Description

The `twitter` function creates a `twitter` object, which represents a Twitter connection.

To establish the connection, you must obtain these required credentials from Twitter:

- Consumer key
- Consumer secret
- Access token
- Access token secret

To obtain these credentials, you must first log in to your Twitter account. Then, fill out the form in Create an application.

After you create a `twitter` object, you can use the object functions to retrieve historical Twitter data with a Twitter connection. Or, you can retrieve other Twitter data by using the Twitter REST API to access other REST API endpoints.

## Creation

### Syntax

```
c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret)
```

#### Description

`c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret)` creates a Twitter connection using the consumer key, consumer secret, access token, and access token secret.

#### Input Arguments

**`consumerkey` — Consumer key**
character vector | string scalar

Consumer key (API key), specified as a character vector or string scalar. To obtain your consumer key, fill out the form in Create an application.

Data Types: `char` | `string`

**`consumersecret` — Consumer secret**
character vector | string scalar

Consumer secret (API secret), specified as a character vector or string scalar. To obtain your consumer secret, fill out the form in Create an application.

Data Types: `char` | `string`

**`accesstoken` — Access token**
character vector | string scalar

Access token, specified as a character vector or string scalar. To obtain your access token, fill out the form in Create an application.

Data Types: `char` | `string`

**`accesstokensecret` — Access token secret**
character vector | string scalar

Access token secret, specified as a character vector or string scalar. To obtain your access token secret, fill out the form in Create an application.

Data Types: `char` | `string`

## Properties

**`Name` — Account name**
character vector

Account name in the Twitter profile, specified as a character vector.

Data Types: `char`

**`ScreenName` — Twitter user name**
character vector

Twitter user name, specified as a character vector.

Example: `@username`

Data Types: `char`

**`MetaData` — Twitter account and profile information**
structure

Twitter account and profile information, specified as a structure.

After creating a Twitter connection, you can access your account and profile information using the `MetaData` property. For example:

```
c.MetaData

ans =

  struct with fields:

                                id: 1.234e+17
                            id_str: '123456789101112141'
                              name: 'Full Name'
                       screen_name: 'username'
                       ...
```

(The values here do not represent real Twitter data.)

Data Types: `struct`

### `StatusCode` — Connection status code

`matlab.net.http.StatusCode` object

Connection status code, specified as a `matlab.net.http.StatusCode` object. When this property has the value `OK`, the Twitter connection is successful.

## Object Functions

| | |
|---|---|
| search | Search for Tweets |
| getdata | Retrieve Twitter data |
| postdata | Post Twitter data |

## Examples

### Search for Tweets

Use a Twitter connection object to search for Tweets.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Search for Tweets using the Twitter connection object and the search term `MathWorks`.

```
tweetquery = 'MathWorks';
d = search(c,tweetquery)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access MathWorks® Tweets. Display the 12th Tweet.

```
 d.Body.Data.statuses{12}.text

ans =

    'MATLAB Control Systems Examples https://t.co/g2P86srv33'
```

You can search for other Tweets using the `search` function. To retrieve other Twitter data, use the `getdata` function.

## See Also

`matlab.net.http.ResponseMessage` | `matlab.net.http.StatusCode`

**Topics**

“Conduct Sentiment Analysis Using Historical Tweets” on page 7-2
“Tweet Based on Retrieved Twitter Data” on page 7-6

**External Websites**

Create an application
Twitter REST API Endpoint Reference Documentation

**Introduced in R2017b**

# search

Search for Tweets

## Syntax

```
d = search(c,tweetquery)

d = search(c,tweetquery,parameters)
d = search(c,tweetquery,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

## Description

`d = search(c,tweetquery)` searches Tweets for the term `tweetquery`.

`d = search(c,tweetquery,parameters)` searches Tweets using web service query parameters. The Twitter REST API defines web service query parameters. For valid parameters, see GET search/tweets.

`d = search(c,tweetquery,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` specifies web service query parameters as one or more pairs of name-value arguments.

## Examples

### Search for Tweets

Use a Twitter connection object to search for Tweets.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode

ans =

    OK
```

Search for Tweets using the Twitter connection object and the search term `MathWorks`.

```
tweetquery = 'MathWorks';
d = search(c,tweetquery)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access MathWorks Tweets. Display the 12th Tweet.

```
d.Body.Data.statuses{12}.text
```

```
ans =

    'MATLAB Control Systems Examples https://t.co/g2P86srv33'
```

You can search for other Tweets using the `search` function. To retrieve other Twitter data, use the `getdata` function.

### Search for Specific Number of Tweets Using Structure

Use a Twitter connection object to search for 20 Tweets. Specify the number of Tweets to retrieve as a structure.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Specify the search term `MathWorks` in the variable `tweetquery`. Specify 20 Tweets as a field in the structure `parameters`. Search for 20 Tweets using the Twitter connection object, search term `tweetquery`, and structure `parameters`.

```
tweetquery = 'MathWorks';
parameters.count = 20;
d = search(c,tweetquery,parameters)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access MathWorks Tweets. Display the structure `Data`.

```
d.Body.Data
```

```
ans =

  struct with fields:

           statuses: {20×1 cell}
    search_metadata: [1×1 struct]
```

The structure `Data` contains the field `statuses`. This field is a cell array of structures. Each structure in the cell array contains information about one Tweet.

Access all 20 Tweets.

```
d.Body.Data.statuses{:}
```

```
ans =

  struct with fields:

                  created_at: 'Fri Apr 28 17:51:55 +0000 2017'
                          id: 1.2345e+17
                      id_str: '123456789101112131'
                        text: 'This collection of over 400 MATLAB examples can help you with #controlsystems, Kalman filters, and more
                   truncated: 0
                    entities: [1×1 struct]
                    metadata: [1×1 struct]
                      source: 'Twitter for iPhone'
       in_reply_to_status_id: []
    in_reply_to_status_id_str: []
          in_reply_to_user_id: []
      in_reply_to_user_id_str: []
      in_reply_to_screen_name: []
                         user: [1×1 struct]
                          geo: []
                  coordinates: []
                        place: []
                 contributors: []
            retweeted_status: [1×1 struct]
             is_quote_status: 0
               retweet_count: 34
              favorite_count: 0
                   favorited: 0
                   retweeted: 0
           possibly_sensitive: 0
                         lang: 'en'
...
```

The field `text` in each structure contains the text of one Tweet.

(These values do not represent real Twitter data.)

You can search for other Tweets using the `search` function. To retrieve other Twitter data, use the `getdata` function.

### Search for Specific Number of Tweets Using Name-Value Arguments

Use a Twitter connection object to search for 20 Tweets. Specify the number of Tweets to retrieve as a name-value argument.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Search for 20 Tweets using the Twitter connection object, search term `MathWorks`, and name-value argument `count`.

```
tweetquery = 'MathWorks';
d = search(c,tweetquery,'count',20)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access MathWorks Tweets. Display the structure `Data`.

```
 d.Body.Data
```

```
ans =

  struct with fields:

           statuses: {20×1 cell}
    search_metadata: [1×1 struct]
```

The structure `Data` contains the field `statuses`. This field is a cell array of structures. Each structure in the cell array contains information about one Tweet.

Access all 20 Tweets.

```
d.Body.Data.statuses{:}
```

```
ans =

  struct with fields:

                  created_at: 'Fri Apr 28 17:51:55 +0000 2017'
                          id: 1.2345e+17
                      id_str: '123456789101112131'
                        text: 'This collection of over 400 MATLAB examples can help you with #controlsystems, Kalman filters, and more
                   truncated: 0
                    entities: [1×1 struct]
                    metadata: [1×1 struct]
                      source: 'Twitter for iPhone'
       in_reply_to_status_id: []
   in_reply_to_status_id_str: []
         in_reply_to_user_id: []
     in_reply_to_user_id_str: []
     in_reply_to_screen_name: []
                        user: [1×1 struct]
                         geo: []
                coordinates: []
                       place: []
                contributors: []
            retweeted_status: [1×1 struct]
             is_quote_status: 0
               retweet_count: 34
              favorite_count: 0
                   favorited: 0
                   retweeted: 0
          possibly_sensitive: 0
                        lang: 'en'
...
```

The field `text` in each structure contains the text of one Tweet.

(These values do not represent real Twitter data.)

You can search for other Tweets using the `search` function. To retrieve other Twitter data, use the `getdata` function.

## Input Arguments

### `c` — Twitter connection

`twitter` object

Twitter connection, specified as a `twitter` object.

### `tweetquery` — Tweet search term

character vector | string scalar

Tweet search term, specified as a character vector or string scalar.

Example: `"MathWorks"`

Data Types: `char` | `string`

### `parameters` — Web service query parameters

structure

Web service query parameters, specified as a structure. Each parameter is specified as a field in the structure. Set the field to a specific value in the structure. For example, specify the number of Tweets to retrieve:

```
parameters.count = 20;
```

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see GET search/tweets.

Data Types: `struct`

**`QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters**
name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of a query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see GET search/tweets.

Example: `'count',20` retrieves 20 Tweets.

Data Types: `char` | `string`

## Output Arguments

**`d` — Twitter data**
`matlab.net.http.ResponseMessage`

Twitter data, returned as a `matlab.net.http.ResponseMessage` object.

To retrieve Twitter data, access properties in `d`, for example:

```
data = d.Body.Data
```

```
data =

      struct with fields:

               statuses: {50×1 cell}
        search_metadata: [1×1 struct]
```

Continue to access the nested structure `data` to retrieve Twitter data. For accessing nested structures, see “Access Data in Nested Structures”.

## Limitations

- The Twitter REST API `GET search/tweets` endpoint specifies that you can retrieve up to a maximum of 100 Tweets at a time.
- The Twitter REST API `GET search/tweets` endpoint specifies that you can retrieve up to 7 days of historical Tweets.

## See Also

**Functions**
`getdata` | `postdata`

**Objects**
`twitter`

**Topics**
“Conduct Sentiment Analysis Using Historical Tweets” on page 7-2

**External Websites**
GET search/tweets
Twitter REST API Endpoint Reference Documentation

**Introduced in R2017b**

# getdata

Retrieve Twitter data

## Syntax

```
d = getdata(c,baseurl)
d = getdata(c,baseurl,parameters)
d = getdata(c,baseurl,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

## Description

`d = getdata(c,baseurl)` retrieves Twitter data for REST API GET endpoints that do not require any web service query parameters.

`d = getdata(c,baseurl,parameters)` retrieves Twitter data using web service query parameters. The Twitter REST API defines web service query parameters for each endpoint. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

`d = getdata(c,baseurl,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` specifies web service query parameters as one or more pairs of name-value arguments.

## Examples

### Retrieve Twitter Data Without Specifying Parameters

Use a Twitter connection object to return locations for trending topics. The REST API endpoint `GET trends/available` does not require any web service query parameters.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Specify the Twitter base URL.

```
baseurl = 'https://api.twitter.com/1.1/trends/available.json';
```

Retrieve locations for trending topics using the Twitter connection object and the base URL.

```
d = getdata(c,baseurl)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access the location data. Display the structure `Data`.

```
d.Body.Data
```

```
ans =

  467×1 struct array with fields:

    name
    placeType
    url
    parentid
    country
    woeid
    countryCode
```

The structure `Data` is a structure array with the field `name`, which contains the name of a location for a trending topic.

Access the first location.

```
d.Body.Data(1).name
```

```
ans =

    'Worldwide'
```

You can retrieve data for other REST API endpoints by substituting another URL for the `baseurl` input argument. Or, you can search for Tweets using the `search` function.

### Retrieve Twitter Data Using Structure

Use a Twitter connection object to retrieve follower information. Specify the count of followers as a structure.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
```

```
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Set the Twitter base URL to access the `GET followers/list` REST API endpoint. Specify one follower by defining the structure `parameters` with the field set to 1. Search for one follower of the current account using the Twitter connection object, base URL, and structure `parameters`.

```
baseurl = 'https://api.twitter.com/1.1/followers/list.json';
parameters.count = 1;
d = getdata(c,baseurl,parameters)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access information about the follower.

```
d.Body.Data.users
```

```
ans =

  struct with fields:

                                    id: 12345678
                                id_str: '12345678'
                                  name: 'Full Name'
...
```

`d.Body.Data.users` is a structure that has a field for each piece of account information. For example, the first three fields are:

- Account identifier as a number
- Account identifier as a character vector
- Full name of the account as a character vector

(These values do not represent real Twitter data.)

You can retrieve data for other REST API endpoints by substituting another URL for the `baseurl` input argument. Or, you can search for Tweets using the `search` function.

### Retrieve Twitter Data Using Name-Value Arguments

Use a Twitter connection object to retrieve follower information. Specify the count of followers as a name-value argument.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Set the Twitter base URL to access the `GET followers/list` REST API endpoint. Search for one follower of the current account using the Twitter connection object, base URL, and name-value argument `count`.

```
baseurl = 'https://api.twitter.com/1.1/followers/list.json';
d = getdata(c,baseurl,'count',1)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×25 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access information about the follower.

```
d.Body.Data.users
```

```
ans =

  struct with fields:
```

```
        id: 12345678
    id_str: '12345678'
      name: 'Full Name'
...
```

`d.Body.Data.users` is a structure that has a field for each piece of account information. For example, the first three fields are:

- Account identifier as a number
- Account identifier as a character vector
- Full name of the account as a character vector

(These values do not represent real Twitter data.)

You can retrieve data for other REST API endpoints by substituting another URL for the `baseurl` input argument. Or, you can search for Tweets using the `search` function.

## Input Arguments

### `c` — Twitter connection

`twitter` object

Twitter connection, specified as a `twitter` object.

### `baseurl` — Twitter base URL

character vector | string scalar

Twitter base URL, specified as a character vector or string scalar. Use this URL to access the Twitter REST API endpoints.

Example: `'https://api.twitter.com/1.1/followers/list.json'` specifies a GET REST API endpoint.

Data Types: `char` | `string`

### `parameters` — Web service query parameters

structure

Web service query parameters, specified as a structure. Each parameter is specified as a field in the structure. Set the field to a specific value in the structure. For example, specify the number of items for the HTTP request:

```
parameters.count = 20;
```

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

Data Types: `struct`

### `QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters

name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of a query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

Example: `'count',20` specifies the number of items for the HTTP request.

Data Types: `char` | `string`

## Output Arguments

### d — Twitter data

`matlab.net.http.ResponseMessage`

Twitter data, returned as a `matlab.net.http.ResponseMessage` object.

To retrieve Twitter data, access properties in `d`, for example:

```
data = d.Body.Data
```

```
data =

      struct with fields:

               statuses: {50×1 cell}
        search_metadata: [1×1 struct]
```

Continue to access the nested structure `data` to retrieve Twitter data. For accessing nested structures, see “Access Data in Nested Structures”.

## Limitations

- Each Twitter REST GET API endpoint has its own limitations. For details, see the Twitter REST API Endpoint Reference Documentation.

## See Also

### Functions

`postdata` | `search`

### Objects

`twitter`

### Topics

“Tweet Based on Retrieved Twitter Data” on page 7-6

### External Websites

Twitter REST API Endpoint Reference Documentation

### Introduced in R2017b

# postdata

Post Twitter data

## Syntax

```
d = postdata(c,baseurl)
d = postdata(c,baseurl,parameters)
d = postdata(c,baseurl,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

## Description

`d = postdata(c,baseurl)` posts Twitter data for REST API POST endpoints that do not require any web service query parameters.

`d = postdata(c,baseurl,parameters)` posts Twitter data using web service query parameters. The Twitter REST API defines web service query parameters for each endpoint. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

`d = postdata(c,baseurl,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` specifies web service query parameters as one or more pairs of name-value arguments.

## Examples

### Post Twitter Data Without Specifying Parameters

Use a Twitter connection object to check Twitter account settings. The REST API endpoint `POST account/settings` does not require any web service query parameters.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Specify the Twitter base URL.

```
baseurl = 'https://api.twitter.com/1.1/account/settings.json';
```

Retrieve account settings using the Twitter connection object and base URL.

```
d = postdata(c,baseurl)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×22 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access account settings data. Display the structure `Data`.

```
d.Body.Data
```

```
ans =

  struct with fields:

                       protected: 0
                     screen_name: 'screenName'
                  always_use_https: 1
       use_cookie_personalization: 0
                      sleep_time: [1×1 struct]
                     geo_enabled: 0
                        language: 'en'
            discoverable_by_email: 0
      discoverable_by_mobile_phone: 0
           display_sensitive_media: 0
         allow_contributor_request: 'none'
                  allow_dms_from: 'following'
             allow_dm_groups_from: 'following'
                  translator_type: 'none'
```

(These values do not represent real Twitter data.)

You can post data using other REST API endpoints by substituting another URL for the `baseurl` input argument.

### Post Twitter Data Using Structure

Use a Twitter connection object to create a Twitter search. Specify the search term for the saved search using a structure.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
```

```
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Specify the search term `MathWorks` as a field of the structure `parameters`. Specify the Twitter base URL for the REST API POST endpoint `POST saved_searches/create`.

```
parameters.query = 'MathWorks';
baseurl = 'https://api.twitter.com/1.1/saved_searches/create.json';
```

Create a saved search using the Twitter connection object, base URL, and structure `parameters`.

```
d = postdata(c,baseurl,parameters)
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×23 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access the saved search data.

```
d.Body.Data
```

```
ans =

  struct with fields:

              id: 8.6011e+17
       id_str: '860112019273416704'
        query: 'MathWorks'
         name: 'MathWorks'
     position: []
    created_at: 'Thu May 04 12:41:00 +0000 2017'
```

`d.Body.Data` is a structure that contains information about the saved search in fields. For example, the field `query` contains the search term `MathWorks` as a character vector.

You can post data using other REST API endpoints by substituting another URL for the `baseurl` input argument.

### Post Twitter Data Using Name-Value Arguments

Use a Twitter connection object to create a Twitter search. Specify the search term for the saved search as a name-value argument.

Create a Twitter connection using your credentials. (The values in this example do not represent real Twitter credentials.)

```
consumerkey = 'abcdefghijklmnop123456789';
consumersecret = 'qrstuvwxyz123456789';
accesstoken = '123456789abcdefghijklmnop';
accesstokensecret = '123456789qrstuvwxyz';

c = twitter(consumerkey,consumersecret,accesstoken,accesstokensecret);
```

Check the Twitter connection. If the `StatusCode` property has the value `OK`, the connection is successful.

```
c.StatusCode
```

```
ans =

    OK
```

Specify the Twitter base URL for the REST API POST endpoint `POST saved_searches/create`.

```
baseurl = 'https://api.twitter.com/1.1/saved_searches/create.json';
```

Create a saved search for the search term `MathWorks` using the Twitter connection object, base URL, and name-value argument `query`.

```
d = postdata(c,baseurl,'query','MathWorks')
```

```
d =

  ResponseMessage with properties:

    StatusLine: 'HTTP/1.1 200 OK'
    StatusCode: OK
        Header: [1×23 matlab.net.http.HeaderField]
          Body: [1×1 matlab.net.http.MessageBody]
     Completed: 0
```

`d` is a `matlab.net.http.ResponseMessage` object. The `StatusCode` property shows `OK`, indicating a successful HTTP request.

Access the saved search data.

```
d.Body.Data
```

```
ans =

  struct with fields:

              id: 8.6011e+17
          id_str: '860112019273416704'
           query: 'MathWorks'
```

```
          name: 'MathWorks'
      position: []
    created_at: 'Thu May 04 12:41:00 +0000 2017'
```

`d.Body.Data` is a structure that contains information about the saved search in fields. For example, the field `query` contains the search term `MathWorks` as a character vector.

You can post data using other REST API endpoints by substituting another URL for the `baseurl` input argument.

## Input Arguments

### `c` — Twitter connection

`twitter` object

Twitter connection, specified as a `twitter` object.

### `baseurl` — Twitter base URL

character vector | string scalar

Twitter base URL, specified as a character vector or string scalar. Use this URL to access the Twitter REST API endpoints.

Example: `'https://api.twitter.com/1.1/followers/list.json'` specifies a GET REST API endpoint.

Data Types: `char` | `string`

### `parameters` — Web service query parameters

structure

Web service query parameters, specified as a structure. Each parameter is specified as a field in the structure. Set the field to a specific value in the structure. For example, specify the number of items for the HTTP request:

```
parameters.count = 20;
```

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

Data Types: `struct`

### `QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters

name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of a query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

The Twitter REST API defines web service query parameters that it accepts as part of an HTTP request. For valid parameters, see the Twitter REST API Endpoint Reference Documentation.

Example: `'count',20` specifies the number of items for the HTTP request.

Data Types: `char` | `string`

## Output Arguments

### d — Twitter data

`matlab.net.http.ResponseMessage`

Twitter data, returned as a `matlab.net.http.ResponseMessage` object.

To retrieve Twitter data, access properties in `d`, for example:

```
data = d.Body.Data
```

```
data =

      struct with fields:

               statuses: {50×1 cell}
         search_metadata: [1×1 struct]
```

Continue to access the nested structure `data` to retrieve Twitter data. For accessing nested structures, see “Access Data in Nested Structures”.

## Limitations

- Each Twitter REST POST API endpoint has its own limitations. For details, see the Twitter REST API Endpoint Reference Documentation.

## See Also

### Functions

`getdata` | `search`

### Objects

`twitter`

### Topics

“Tweet Based on Retrieved Twitter Data” on page 7-6

### External Websites

Twitter REST API Endpoint Reference Documentation

### Introduced in R2017b

# trth

Tick History from Refinitiv connection

## Description

The `trth` function creates a `trth` object, which represents a Tick History from Refinitiv connection. This connection uses the Tick History REST API to retrieve data. After you create a `trth` object, you can use the object functions to retrieve historical and intraday data.

## Creation

### Syntax

```
c = trth(username,password)
```

#### Description

`c = trth(username,password)` creates a Tick History from Refinitiv connection object using a password and sets the Username property.

#### Input Arguments

**`password` — Password**
character vector | string scalar

Password, specified as a character vector or string scalar. For credentials, contact Refinitiv.

Example: `'password'`

Data Types: `char` | `string`

### Properties

**`Username` — User name**
character vector | string scalar

User name, specified as a character vector or string scalar. For credentials, contact Refinitiv.

Example: `'username'`

Data Types: `char` | `string`

**`TimeOut` — Timeout**
`200` (default) | numeric scalar

Timeout value in seconds that MATLAB attempts to connect by using the Tick History REST API, specified as a numeric scalar.

Example: `100`

Data Types: `double`

## Object Functions

| | |
|---|---|
| history | Tick History from Refinitiv historical data |
| timeseries | Tick History from Refinitiv intraday data |

## Examples

### Retrieve Historical Data for One Security

Use a Tick History connection to retrieve historical data for a security.

Create a Tick History connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve historical data for the IBM security. Using the `history` function, retrieve the open and closing prices for the prior day.

```
sec = ["IBM.N","Ric"];
fields = ["Open";"Last"];
startdate = datetime('yesterday');
enddate = datetime('today');

d = history(c,sec,fields,startdate,enddate)
```

```
d =

  1×2 timetable

      Time         Open      Last
    ___________    ______    ______

    2017/11/02     154.25    153.35
```

`d` is a timetable that contains these variables:

- Date for the prior day
- Open price
- Closing price

## See Also

### Topics

“Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv” on page 8-2
“Decide to Sell Shares with Historical Data Using Tick History from Refinitiv” on page 8-4

### External Websites

Refinitiv REST API Documentation

### Introduced in R2018a

# history

Tick History from Refinitiv historical data

## Syntax

```
d = history(c,sec,fields,startdate,enddate)
```

## Description

`d = history(c,sec,fields,startdate,enddate)` returns historical data using:

- Tick History from Refinitiv connection object
- Refinitiv securities (for example, Reuters Instrument Codes, or RICs)
- Refinitiv historical fields
- Start date for the beginning of the historical date range
- End date for the end of the historical date range

## Examples

### Retrieve Historical Data for One Security

Use a Tick History connection to retrieve historical data for a security.

Create a Tick History connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve historical data for the IBM security. Using the `history` function, retrieve the open and closing prices for the prior day.

```
sec = ["IBM.N","Ric"];
fields = ["Open";"Last"];
startdate = datetime('yesterday');
enddate = datetime('today');

d = history(c,sec,fields,startdate,enddate)
```

```
d =

  1×2 timetable

      Time         Open      Last
    ___________    ______    ______

    2017/11/02     154.25    153.35
```

`d` is a timetable that contains these variables:

- Date for the prior day
- Open price
- Closing price

### Retrieve Historical Data for Two Securities

Use a Tick History connection to retrieve historical data for two securities.

Create a Tick History from Refinitiv connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve historical data for the IBM and Ford Motor Company securities. Using the `history` function, retrieve the open and closing prices for the days in the date range from October 30, 2017 through November 3, 2017.

```
sec = ["IBM.N","Ric";"F.N","Ric"];
fields = ["Open";"Last"];
startdate = datetime('10/30/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/03/2017','InputFormat','MM/dd/yyyy');

d = history(c,sec,fields,startdate,enddate)
```

```
d =

  10×2 timetable

       Time         Open      Last
    ____________    ______    ______

    2017/10/30     153.76    154.36
    2017/10/31     154.26    154.06
    2017/11/01     153.97    154.03
    2017/11/02     154.25    153.35
    2017/11/03     153.36    151.58
    2017/10/30      12.00     12.10
    2017/10/31      12.14     12.27
    2017/11/01      12.40     12.35
    2017/11/02      12.33     12.42
    2017/11/03      12.40     12.36
```

`d` is a timetable that contains these variables:

- Date in the date range
- Open price
- Closing price

The first five rows contain data for the IBM security. The next five rows contain data for the Ford Motor Company security.

## Input Arguments

### `c` — Tick History from Refinitiv connection

connection object

Tick History from Refinitiv connection, specified as a connection object created with `trth`.

### `sec` — Security

string array | cell array of character vectors

Security, specified as an N-by-2 string array or cell array of character vectors. The first column of the string array or cell array identifies the security. The second column identifies the type of security (for example, Reuters Instrument Code, or RIC).

Example: `["IBM.N","Ric"]`

Data Types: `string` | `cell`

### `fields` — Fields

string array | cell array of character vectors

Fields, specified as a string array or cell array of character vectors. Specify Refinitiv historical fields to retrieve historical data. You can search for fields in the Refinitiv Customer Zone.

Example: `["Low";"Last";"Volume"]`

Data Types: `string` | `cell`

### `startdate` — Start date

`datetime` array | string scalar | character vector | numeric scalar

Start date of the date range, specified as a `datetime` array, string scalar, character vector, or numeric scalar.

Example: `datetime('yesterday')`

Data Types: `double` | `char` | `string` | `datetime`

### `enddate` — End date

`datetime` array | string scalar | character vector | numeric scalar

End date of the date range, specified as a `datetime` array, string scalar, character vector, or numeric scalar.

Example: `datetime('today')`

Data Types: `double` | `char` | `string` | `datetime`

## Output Arguments

### `d` — Historical data

timetable

Historical data from Tick History, returned as a timetable. Except for the `Time` variable, each variable in the timetable corresponds to a specified field in the `fields` input argument.

## See Also

`timeseries` | `trth`

**Topics**

“Decide to Sell Shares with Historical Data Using Tick History from Refinitiv” on page 8-4

**External Websites**

Refinitiv REST API Documentation

**Introduced in R2018a**

# timeseries

Tick History from Refinitiv intraday data

## Syntax

```
d = timeseries(c,sec,fields,startdate,enddate)
d = timeseries(c,sec,fields,startdate,enddate,interval)
```

## Description

`d = timeseries(c,sec,fields,startdate,enddate)` returns intraday data using:

- Tick History from Refinitiv connection object
- Refinitiv securities (for example, Reuters Instrument Codes, or RICs)
- Refinitiv intraday fields
- Start date for the beginning of the intraday date range
- End date for the end of the intraday date range

`d = timeseries(c,sec,fields,startdate,enddate,interval)` uses an aggregation value for the intraday data.

## Examples

### Retrieve Intraday Data for One Security

Use a Tick History connection to retrieve intraday data for one security.

Create a Tick History from Refinitiv connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve intraday data for the IBM security. Using the `timeseries` function, retrieve the exchange time, price, and volume from November 6, 2017 through November 7, 2017.

```
sec = ["IBM.N","Ric"];
fields = ["Trade - Exchange Time";"Trade - Price";"Trade - Volume"];
startdate = datetime('11/06/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/07/2017','InputFormat','MM/dd/yyyy');

d = timeseries(c,sec,fields,startdate,enddate);
```

Display the first three rows of intraday data.

```
head(d,3)
```

```
ans =
```

```
3×7 timetable

          Time              x_RIC         Domain        GMTOffset     Type       Price      Volu
    ____________________    _____    ________________    _________    _____    ________    ____

    06-Nov-2017 05:31:02    'IBM.N'    'Market Price'       '-5'       'Trade'    ''           Na
    06-Nov-2017 14:30:10    'IBM.N'    'Market Price'       '-5'       'Trade'    '151.68'    698
    06-Nov-2017 14:30:10    'IBM.N'    'Market Price'       '-5'       'Trade'    '151.66'
```

`d` is a timetable that contains these variables:

- Transaction date and time
- RIC
- Domain
- GMT time zone offset
- Transaction type
- Price
- Volume
- Exchange time

### Retrieve Aggregated Intraday Data for Two Securities

Use a Tick History connection to retrieve intraday data for two securities, aggregated into 1-hour intervals.

Create a Tick History from Refinitiv connection by using a user name and password. The appearance of the connection object `c` in the MATLAB workspace indicates a successful connection.

```
username = 'username';
password = 'password';
c = trth(username,password);
```

Retrieve intraday data for the IBM and Ford Motor Company securities. Using the `timeseries` function, retrieve the open, high, low, and last prices from November 6, 2017 through November 7, 2017. Aggregate the intraday data into 1-hour intervals.

```
sec = ["IBM.N","Ric";"F.N","Ric"];
fields = ["Open";"High";"Low";"Last"];
startdate = datetime('11/06/2017','InputFormat','MM/dd/yyyy');
enddate = datetime('11/07/2017','InputFormat','MM/dd/yyyy');
interval = 'OneHour';

d = timeseries(c,sec,fields,startdate,enddate,interval);
```

`d` is a timetable that contains 66 rows of aggregated intraday data. The first 33 rows contain data for the Ford Motor Company security. The last 33 rows contain data for the IBM security.

Display three rows of Ford Motor Company aggregated intraday data.

```
d(15:17,:)

ans =
```

```
  3×8 timetable

          Time            x_RIC        Domain        GMTOffset          Type          Open
    ____________________    _____    _____________    _________    _________________    _____

    06-Nov-2017 14:00:00    'F.N'    'Market Price'      '-5'       'Intraday 1Hour'    12.34
    06-Nov-2017 15:00:00    'F.N'    'Market Price'      '-5'       'Intraday 1Hour'    12.38
    06-Nov-2017 16:00:00    'F.N'    'Market Price'      '-5'       'Intraday 1Hour'    12.32
```

Display three rows of IBM aggregated intraday data.

```
d(48:50,:)
```

```
ans =

  3×8 timetable

          Time            x_RIC        Domain        GMTOffset          Type          Open
    ____________________    _____    _____________    _________    _________________    _____

    06-Nov-2017 14:00:00    'IBM.N'    'Market Price'      '-5'       'Intraday 1Hour'    151.68
    06-Nov-2017 15:00:00    'IBM.N'    'Market Price'      '-5'       'Intraday 1Hour'    151.13
    06-Nov-2017 16:00:00    'IBM.N'    'Market Price'      '-5'       'Intraday 1Hour'    150.78
```

`d` is a timetable that contains these variables:

- Transaction date and time
- RIC
- Domain
- GMT time zone offset
- Aggregation type
- Open price
- High price
- Low price
- Last price

## Input Arguments

### c — Tick History from Refinitiv connection

connection object

Tick History from Refinitiv connection, specified as a connection object created with `trth`.

### sec — Security

string array | cell array of character vectors

Security, specified as an `N`-by-2 string array or cell array of character vectors. The first column of the string array or cell array identifies the security. The second column identifies the type of security (for example, Reuters Instrument Code, or RIC).

Example: `["IBM.N","Ric"]`

Data Types: `string` | `cell`

**`fields` — Fields**
string array | cell array of character vectors

Fields, specified as a string array or cell array of character vectors. Specify Refinitiv intraday fields to retrieve intraday data. You can search for fields in the Refinitiv Customer Zone.

Example: `["Trade - Exchange Time";"Trade - Price"]`

Data Types: `string` | `cell`

**`startdate` — Start date**
`datetime` array | string scalar | character vector | numeric scalar

Start date of the date range, specified as a `datetime` array, string scalar, character vector, or numeric scalar.

Example: `datetime('yesterday')`

Data Types: `double` | `char` | `string` | `datetime`

**`enddate` — End date**
`datetime` array | string scalar | character vector | numeric scalar

End date of the date range, specified as a `datetime` array, string scalar, character vector, or numeric scalar.

Example: `datetime('today')`

Data Types: `double` | `char` | `string` | `datetime`

**`interval` — Aggregation interval**
`'OneSecond'` | `'FiveSeconds'` | `'OneMinute'` | ...

Aggregation interval, specified as one of these values:

- `'OneSecond'`
- `'FiveSeconds'`
- `'OneMinute'`
- `'FiveMinutes'`
- `'TenMinutes'`
- `'FifteenMinutes'`
- `'OneHour'`

Specify these values as character vectors or string scalars.

## Output Arguments

**`d` — Intraday data**
timetable

Intraday data, returned as a timetable with these variables:

- Transaction date and time
- RIC

- Domain
- GMT time zone offset
- Transaction type

Other variables in `d` include the specified fields in the `fields` input argument.

## See Also

`history` | `trth`

**Topics**

“Decide to Buy Shares with Intraday Data Using Tick History from Refinitiv” on page 8-2

**External Websites**

Refinitiv REST API Documentation

**Introduced in R2018a**

# quandl

Quandl connection

## Description

The `quandl` function creates a `quandl` object, which represents a Quandl connection. To establish the connection, you must obtain a Quandl API key from the Quandl website by creating an account.

After you create a `quandl` object, you can use the `history` function to retrieve historical data.

## Creation

### Syntax

```
c = quandl(apikey)
```

#### Description

`c = quandl(apikey)` creates a Quandl connection using a Quandl API key.

#### Input Arguments

**`apikey` — Quandl API key**
character vector | string scalar

Quandl API key, specified as a character vector or string scalar. To obtain your API key, create an account using the Quandl website.

Data Types: `char` | `string`

### Properties

**`TimeOut` — Timeout**
numeric scalar

Timeout, specified as a numeric scalar that indicates the number of seconds to wait for data to return before canceling the request.

Example: 15

Data Types: `double`

### Object Functions

history  Retrieve Quandl historical data

### Examples

### Retrieve Quandl Historical Data

Use a Quandl connection to retrieve historical data for a security within the available date range for the specified security.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey)
```

```
c =

  quandl with properties:

    TimeOut: 100
```

`c` is the `quandl` connection object with the `TimeOut` property. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return historical data before canceling the request.

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security within the available date range for the security. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. The specified security indicates the default periodicity (in this case, daily). `d` is a timetable with the time in the first variable and the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
d = history(c,s);
```

Display the first few rows of historical prices.

```
head(d)
```

```
ans =

  8×1 timetable

      Time        PreviousSettlement
    ___________    __________________

    28-Apr-2018          294.00
    27-Apr-2018          294.00
    26-Apr-2018          294.00
    25-Apr-2018          287.00
    24-Apr-2018          287.00
    23-Apr-2018          289.50
    21-Apr-2018          291.00
    20-Apr-2018          291.00
```

Decide to buy or sell this contract based on the historical prices.

## See Also

**Topics**

“Retrieve Historical Data Using Quandl” on page 1-21

“Access Quandl Error Messages” on page 9-2

**External Websites**
Quandl

**Introduced in R2018b**

# history

Retrieve Quandl historical data

## Syntax

```
d = history(c,s)
d = history(c,s,startdate,enddate)
d = history(c,s,startdate,enddate,periodicity)
d = history(c,s,startdate,enddate,periodicity,
QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

## Description

`d = history(c,s)` retrieves Quandl historical data using a Quandl connection and a security.

`d = history(c,s,startdate,enddate)` also specifies a date range for the historical data to retrieve.

`d = history(c,s,startdate,enddate,periodicity)` also specifies the periodicity for the historical data to retrieve.

`d = history(c,s,startdate,enddate,periodicity,
QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` also specifies web service query parameters as one or more pairs of name-value arguments.

## Examples

### Retrieve Quandl Historical Data

Use a Quandl connection to retrieve historical data for a security within the available date range for the specified security.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey)
```

```
c =

  quandl with properties:

    TimeOut: 100
```

`c` is the `quandl` connection object with the `TimeOut` property. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return historical data before canceling the request.

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security within the available date range for the security. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. The specified security indicates the default periodicity (in this case, daily). `d` is a timetable with the time in the first variable and the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
d = history(c,s);
```

Display the first few rows of historical prices.

```
head(d)
```

```
ans =

  8×1 timetable

        Time        PreviousSettlement
    ____________    __________________

    28-Apr-2018           294.00
    27-Apr-2018           294.00
    26-Apr-2018           294.00
    25-Apr-2018           287.00
    24-Apr-2018           287.00
    23-Apr-2018           289.50
    21-Apr-2018           291.00
    20-Apr-2018           291.00
```

Decide to buy or sell this contract based on the historical prices.

### Retrieve Quandl Historical Data Within Date Range

Use a Quandl connection to retrieve historical data for a security within a specified date range.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security from March 1, 2018 through March 31, 2018. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. The specified security indicates the default periodicity (in this case, daily). `d` is a timetable with the time in the first variable and the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
startdate = datetime('03-01-2018','InputFormat','MM-dd-yyyy');
enddate = datetime('03-31-2018','InputFormat','MM-dd-yyyy');
d = history(c,s,startdate,enddate);
```

Display the first few rows of historical prices.

```
head(d)
```

```
ans =

  8×1 timetable

      Time        PreviousSettlement
    ___________    __________________

    31-Mar-2018          277.50
    30-Mar-2018          277.50
    29-Mar-2018          277.50
    28-Mar-2018          278.50
    27-Mar-2018          278.00
    26-Mar-2018          278.50
    24-Mar-2018          280.00
    23-Mar-2018          280.00
```

Decide to buy or sell this contract based on the historical prices.

### Retrieve Quandl Historical Data Using Periodicity

Use a Quandl connection to retrieve historical data for a security within the specified date range and using the specified periodicity.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security from March 1, 2018 through March 31, 2018. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. Use a weekly period for the returned data. `d` is a timetable with the time in the first variable and the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
startdate = datetime('03-01-2018','InputFormat','MM-dd-yyyy');
enddate = datetime('03-31-2018','InputFormat','MM-dd-yyyy');
periodicity = 'weekly';
d = history(c,s,startdate,enddate,periodicity);
```

Display the historical prices.

```
d
```

```
d =

  5×1 timetable

      Time        PreviousSettlement
    ___________    __________________
```

```
    01-Apr-2018          277.50
    25-Mar-2018          280.00
    18-Mar-2018          281.50
    11-Mar-2018          279.50
    04-Mar-2018          283.00
```

Decide to buy or sell this contract based on the historical prices.

### Retrieve Quandl Historical Data Using Transformation

Use a Quandl connection to retrieve historical data for a security within a specified date range. Return data with a weekly periodicity and price transformation.

Create a Quandl connection using a Quandl API key.

```
apikey = 'abcdef12345';
c = quandl(apikey);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve historical data for the `CHRIS/ASX_WM2` security from March 1, 2018 through March 31, 2018. This security provides historical future prices for Eastern Australian Wheat Futures, Continuous Contract #2. Use a weekly period for the returned data. Transform the historical price data by calculating the row-on-row percent change. Specify the calculation using the name-value argument `"transform"` with the `"rdiff"` value. `d` is a timetable with the time in the first variable and the percent change of the previous settlement price in the second variable.

```
s = 'CHRIS/ASX_WM2';
startdate = datetime('03-01-2018','InputFormat','MM-dd-yyyy');
enddate = datetime('03-31-2018','InputFormat','MM-dd-yyyy');
periodicity = 'weekly';
d = history(c,s,startdate,enddate,periodicity, ...
    "transform","rdiff");
```

Display the percent changes.

```
d
```

```
d =

  4×1 timetable

      Time        PreviousSettlement
    ____________    __________________

    01-Apr-2018          -0.01
    25-Mar-2018          -0.01
    18-Mar-2018           0.01
    11-Mar-2018          -0.01
```

Decide to buy or sell this contract based on the percent changes in the historical prices.

## Input Arguments

### c — Quandl connection

`quandl` object

Quandl connection, specified as a `quandl` object.

### s — Security

character vector | string scalar

Security, specified as a character vector or string scalar.

Example: `"CHRIS/ASX_WM2"`

Data Types: `char` | `string`

### startdate — Start date

`datetime` array | numeric scalar | string scalar | character vector

Start date of the date range, specified as a `datetime` array, numeric scalar, string scalar, or character vector. By default, the start date is the date of the first available historical data for the specified security `s`.

If you specify the `enddate` input argument, then you must specify the `startdate` input argument.

Example: `datetime('03-01-2018','InputFormat','MM-dd-yyyy')`

Example: `737121`

Data Types: `double` | `char` | `string` | `datetime`

### enddate — End date

`datetime` array | numeric scalar | string scalar | character vector

End date of the date range, specified as a `datetime` array, numeric scalar, string scalar, or character vector. By default, the end date is the date of the last available historical data for the specified security `s`.

If you specify the `startdate` input argument, then you must specify the `enddate` input argument.

Example: `datetime('03-31-2018','InputFormat','MM-dd-yyyy')`

Example: `737181`

Data Types: `double` | `char` | `string` | `datetime`

### periodicity — Periodicity

`'daily'` | `'weekly'` | `'monthly'` | ...

Periodicity, specified as one these values:

- `'daily'`
- `'weekly'`
- `'monthly'`

- `'quarterly'`
- `'annual'`

You can specify these values as a character vector or string scalar.

The default periodicity depends on the specified security `s`.

**`QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters**
name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector or string scalar that specifies the name of a query parameter. A `QueryValue` argument is a character vector or string scalar that specifies the value of the query parameter.

This table describes the valid names and values for the name-value arguments. Specify the name using a character vector or string scalar.

| Name | Description | Value | Default Value | Value Data Type |
|---|---|---|---|---|
| `"limit"` | Number of rows to return | *n* | Return all rows of data | character vector, string scalar, or numeric scalar |
| `"column_index"` | Column index of Quandl historical data to return | *n* | Return all columns of Quandl historical data | character vector, string scalar, or numeric scalar |
| `"order"` | Sort order for the dates in the returned data | `"asc"` or `"desc"` | `"desc"` | character vector or string scalar |
| `"transform"` | Applied calculation on the returned data | See the following table for the values and their descriptions | `"none"` | character vector or string scalar |

This table describes the values for the `"transform"` name-value argument.

| Value | Description |
|---|---|
| `"none"` | No calculation on the returned data |
| `"diff"` | Row-on-row change |
| `"rdiff"` | Row-on-row percent change |
| `"rdiff_from"` | Latest value as percent increment |
| `"cumul"` | Cumulative sum |
| `"normalize"` | Scale series starts at 100 |

Example: `"limit",10` limits the returned data to 10 rows.

Example: `"transform","diff"` calculates the row-on-row change for the returned historical data.

Data Types: `char` | `string`

## Output Arguments

### d — Quandl historical data

timetable | `matlab.net.http.ResponseMessage`

Quandl historical data, returned as a timetable or a `matlab.net.http.ResponseMessage` object.

If the historical data request is successful, then the `history` function returns data as a timetable. The timetable contains the time in the first variable. The subsequent variables in the timetable correspond to the returned data. The variables of the returned data depend on the specified security `s`.

If the historical data request is unsuccessful, the `history` function returns the error message in the `matlab.net.http.ResponseMessage` object. To access the error message, see “Access Quandl Error Messages” on page 9-2.

## See Also

`quandl`

### Topics

“Retrieve Historical Data Using Quandl” on page 1-21
“Access Quandl Error Messages” on page 9-2

### External Websites

Quandl

### Introduced in R2018b

# datastreamws

Datastream Web Services from Refinitiv connection

## Description

The `datastreamws` function creates a `datastreamws` object, which represents a Datastream Web Services connection. You must obtain Datastream Web Services credentials. For credentials, contact Datastream Web Services. After you create a `datastreamws` object, you can retrieve historical data.

## Creation

### Syntax

```
c = datastreamws(username,password)
```

#### Description

`c = datastreamws(username,password)` creates a Datastream Web Services connection using a user name and password.

#### Input Arguments

**`username` — User name**
character vector | string scalar

User name, specified as a character vector or string scalar. For your user name, contact Datastream Web Services.

Example: `'ABCDEF'`

Data Types: `char` | `string`

**`password` — Password**
character vector | string scalar

Password, specified as a character vector or string scalar. For your password, contact Datastream Web Services.

Example: `'abcdef12345'`

Data Types: `char` | `string`

### Properties

**`Username` — User name**
character vector

This property is read-only.

User name, specified as a character vector. For your user name, contact Datastream Web Services.

The `username` input argument sets the `Username` property.

Example: `'ABCDEF'`

Data Types: `char` | `string`

**`TimeOut` — Timeout**
100 (default) | numeric scalar

Timeout, specified as a numeric scalar that indicates the number of seconds to wait for data to return before canceling the request.

Example: `50`

Data Types: `double`

## Object Functions

history Retrieve historical data from Datastream Web Services from Refinitiv

## Examples

### Retrieve Historical Data for Security

Use a Datastream Web Services connection to retrieve historical data for the specified security.

Create a Datastream Web Services connection using your user name and password.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password)
```

```
c =

  datastreamws with properties:

    Username: 'ABCDEF'
     TimeOut: 100
```

`c` is the `datastreamws` connection object with the `Username` and `TimeOut` properties. The `Username` property contains the specified user name. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return historical data before canceling the request.

Adjust the display format to display currency.

```
format bank
```

Retrieve historical end-of-day price data for the last year. Specify the `VOD` security. `d` is a timetable that contains the date in the first variable and the end-of-day price in the second variable.

```
sec = 'VOD';
d = history(c,sec);
```

Display the first few prices.

```
head(d)
```

```
ans =

  8×1 timetable

           Time             VOD
    ____________________    ______

    03-May-2017 00:00:00    202.95
    04-May-2017 00:00:00    203.70
    05-May-2017 00:00:00    204.95
    08-May-2017 00:00:00    205.15
    09-May-2017 00:00:00    205.15
    10-May-2017 00:00:00    206.60
    11-May-2017 00:00:00    206.25
    12-May-2017 00:00:00    211.05
```

Use the end-of-day prices to make investment decisions for the `VOD` security.

## See Also

**Topics**
“Retrieve Datastream Web Services Historical Data” on page 10-2
“Access Datastream Web Services Error Messages” on page 10-4

**External Websites**
Datastream Web Services from Refinitiv REST Service

**Introduced in R2018b**

# history

Retrieve historical data from Datastream Web Services from Refinitiv

## Syntax

```
d = history(c,sec)
d = history(c,sec,fields,date)
d = history(c,sec,fields,startdate,enddate)
d = history(c,sec,fields,startdate,enddate,period)
[d,response] = history( ___ )
```

## Description

`d = history(c,sec)` retrieves historical Datastream Web Services data for a specified security for the last year.

`d = history(c,sec,fields,date)` retrieves historical data for specified fields and a specific date.

`d = history(c,sec,fields,startdate,enddate)` retrieves historical data for a date range.

`d = history(c,sec,fields,startdate,enddate,period)` retrieves historical data using a period.

`[d,response] = history( ___ )` returns a `ResponseMessage` object that contains an error message. To access the error message, see “Access Datastream Web Services Error Messages” on page 10-4.

## Examples

### Retrieve Historical Data for Security

Use a Datastream Web Services connection to retrieve historical data for the specified security.

Create a Datastream Web Services connection using your user name and password.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password)
```

```
c =

  datastreamws with properties:

    Username: 'ABCDEF'
     TimeOut: 100
```

`c` is the `datastreamws` connection object with the `Username` and `TimeOut` properties. The `Username` property contains the specified user name. The `TimeOut` property specifies waiting for a maximum of 100 seconds to return historical data before canceling the request.

Adjust the display format to display currency.

```
format bank
```

Retrieve historical end-of-day price data for the last year. Specify the `VOD` security. `d` is a timetable that contains the date in the first variable and the end-of-day price in the second variable.

```
sec = 'VOD';
d = history(c,sec);
```

Display the first few prices.

```
head(d)
```

```
ans =

  8×1 timetable

            Time              VOD
    ____________________    ______

    03-May-2017 00:00:00    202.95
    04-May-2017 00:00:00    203.70
    05-May-2017 00:00:00    204.95
    08-May-2017 00:00:00    205.15
    09-May-2017 00:00:00    205.15
    10-May-2017 00:00:00    206.60
    11-May-2017 00:00:00    206.25
    12-May-2017 00:00:00    211.05
```

Use the end-of-day prices to make investment decisions for the `VOD` security.

### Return Historical Data for Specified Fields and Date

Use a Datastream Web Services connection to retrieve historical data for the specified security, fields, and date.

Create a Datastream Web Services connection using your user name and password. `c` is the `datastreamws` connection object.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve and display historical end-of-day price data for March 29, 2018. Specify the `VOD` security and these fields:

- Opening price
- High price
- Last closing price

`d` is a timetable with the date in the first variable and the fields in the subsequent variables.

```
sec = "VOD";
fields = ["PO";"PH";"P"];
date = datetime('03-29-2018','InputFormat','MM-dd-yyyy');
d = history(c,sec,fields,date)
```

```
d =

  1×3 timetable

            Time              PO        PH        P
    ____________________    ______    ______    ______

    29-Mar-2018 00:00:00    194.94    196.01    194.22
```

Use the end-of-day prices for each field to make investment decisions for the `VOD` security.

### Return Historical Data for Date Range

Use a Datastream Web Services connection to retrieve historical data for the specified security, fields, and date range.

Create a Datastream Web Services connection using your user name and password. `c` is the `datastreamws` connection object.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve historical end-of-day price data from April 1, 2018 through April 30, 2018. Specify the `VOD` security and these fields:

- Opening price
- High price
- Last closing price

`d` is a timetable with the date in the first variable and the fields in the subsequent variables.

```
sec = "VOD";
fields = ["PO";"PH";"P"];
startdate = datetime('04-01-2018','InputFormat','MM-dd-yyyy');
enddate = datetime('04-30-2018','InputFormat','MM-dd-yyyy');
d = history(c,sec,fields,startdate,enddate);
```

Display the first few prices.

```
head(d)
```

```
ans =

  8×3 timetable

            Time              PO        PH        P
```

```
    ____________________    ______    ______    ______

    02-Apr-2018 00:00:00       NaN       NaN    194.22
    03-Apr-2018 00:00:00    193.70    194.15    193.90
    04-Apr-2018 00:00:00    196.64    198.10    197.22
    05-Apr-2018 00:00:00    200.45    203.90    203.65
    06-Apr-2018 00:00:00    203.15    205.15    204.00
    09-Apr-2018 00:00:00    204.35    205.45    203.65
    10-Apr-2018 00:00:00    204.45    205.90    205.60
    11-Apr-2018 00:00:00    205.50    207.70    206.30
```

Use the end-of-day prices for each field to make investment decisions for the `VOD` security.

### Return Historical Data Using Period

Use a Datastream Web Services connection to retrieve historical data for the specified security, fields, date range, and period.

Create a Datastream Web Services connection using your user name and password. `c` is the `datastreamws` connection object.

```
username = 'ABCDEF';
password = 'abcdef12345';
c = datastreamws(username,password);
```

Adjust the display format to display currency.

```
format bank
```

Retrieve and display historical end-of-day price data from January 1, 2017 through December 31, 2017. Specify the `VOD` security and these fields:

- Opening price
- High price
- Last closing price

Specify a quarterly period. `d` is a timetable with the date in the first variable and the fields in the subsequent variables.

```
sec = "VOD";
fields = ["PO";"PH";"P"];
startdate = datetime('01-01-2017','InputFormat','MM-dd-yyyy');
enddate = datetime('12-31-2017','InputFormat','MM-dd-yyyy');
period = 'Q';
d = history(c,sec,fields,startdate,enddate,period)
```

```
d =

  4×3 timetable

           Time              PO        PH        P
    ____________________    ______    ______    ______

    01-Jan-2017 00:00:00       NaN       NaN    199.85
    01-Apr-2017 00:00:00    209.00    209.10    206.65
```

```
01-Jul-2017 00:00:00    217.65    219.20    218.70
01-Oct-2017 00:00:00    209.35    211.60    210.50
```

Use the quarterly prices for each field to make investment decisions for the `VOD` security.

## Input Arguments

### c — Datastream Web Services connection

`datastreamws` object

Datastream Web Services connection, specified as a `datastreamws` object.

### sec — Security

character vector | cell array of character vectors | string scalar | string array

Security, specified as a character vector, cell array of character vectors, string scalar, or string array. Use a character vector or string scalar to specify one security. Use a cell array of character vectors or string array to specify multiple securities. For a constituent list, specify a single security in the `sec` input argument, for example, `"LFTSE100"`.

Example: `"VOD"`

Data Types: `char` | `string` | `cell`

### fields — Field list

character vector | cell array of character vectors | string scalar | string array

Field list, specified as a character vector, cell array of character vectors, string scalar, or string array. Use a character vector or string scalar to specify one field. Use a cell array of character vectors or string array to specify multiple fields.

Example: `["PH","PO","P"]`

Data Types: `char` | `string` | `cell`

### date — Date

`datetime` array | numeric scalar | string scalar | character vector

Date, specified as a `datetime` array, numeric scalar, string scalar, or character vector. Use this date to retrieve historical data for a specific day.

Example: `datetime('03-29-2018','InputFormat','MM-dd-yyyy')`

Data Types: `double` | `char` | `string` | `datetime`

### startdate — Start date

`datetime` array | numeric scalar | string scalar | character vector

Start date of a date range, specified as a `datetime` array, numeric scalar, string scalar, or character vector. The default start date is the first date of available historical data for the specified security `sec`.

Example: `datetime('04-01-2018','InputFormat','MM-dd-yyyy')`

Data Types: `double` | `char` | `string` | `datetime`

### enddate — End date

`datetime` array | numeric scalar | string scalar | character vector

End date of a date range, specified as a `datetime` array, numeric scalar, string scalar, or character vector. The default end date is the latest date of available historical data for the specified security `sec`.

Example: `datetime('04-30-2018','InputFormat','MM-dd-yyyy')`

Data Types: `double` | `char` | `string` | `datetime`

**`period` — Period**
`'D'` | `'W'` | `'M'` | `'Q'` | `'Y'`

Period, specified as one of these values:

- `'D'` — Daily
- `'W'` — Weekly
- `'M'` — Monthly
- `'Q'` — Quarterly
- `'Y'` — Yearly

You can specify the value as a character vector or string scalar. The default period depends on the specified security `sec`.

## Output Arguments

**`d` — Historical data**
timetable | table

Historical data, returned as a timetable or table. The `history` function returns a timetable with data for one security. For multiple securities, the `history` function returns a timetable for the first syntax only and a table of nested timetables for the other syntaxes. To access one of the nested timetables, use dot notation, for example, `d.VOD`.

**`response` — Response message**
`matlab.net.http.ResponseMessage`

Response message, returned as a `matlab.net.http.ResponseMessage` object. The `ResponseMessage` object contains an error message. To access the error message, see “Access Datastream Web Services Error Messages” on page 10-4.

## See Also
`datastreamws`

**Topics**
“Retrieve Datastream Web Services Historical Data” on page 10-2
“Access Datastream Web Services Error Messages” on page 10-4

**External Websites**
Datastream Web Services from Refinitiv REST Service

**Introduced in R2018b**