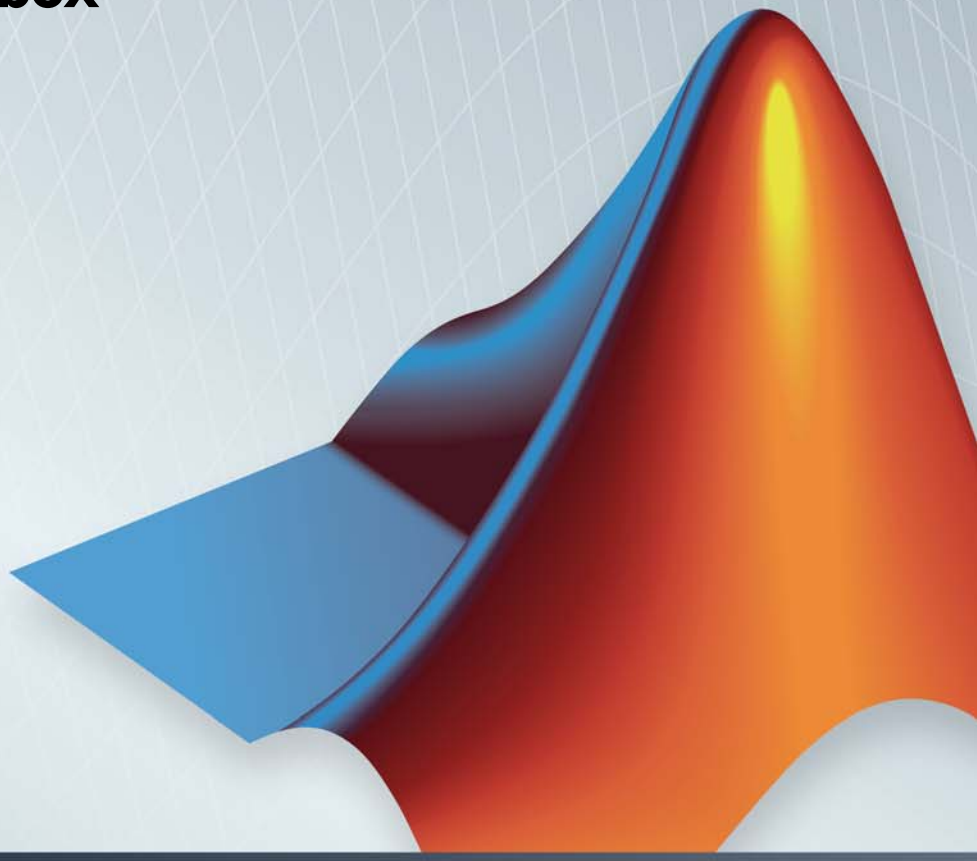
# Datafeed Toolbox™

User's Guide

**R**2014**a**



MATLAB®

MathWorks®

### How to Contact MathWorks

| | |
|---|---|
| `www.mathworks.com` | Web |
| `comp.soft-sys.matlab` | Newsgroup |
| `www.mathworks.com/contact_TS.html` | Technical Support |

| | |
|---|---|
| `suggest@mathworks.com` | Product enhancement suggestions |
| `bugs@mathworks.com` | Bug reports |
| `doc@mathworks.com` | Documentation error reports |
| `service@mathworks.com` | Order status, license renewals, passcodes |
| `info@mathworks.com` | Sales, pricing, and general information |

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

*Datafeed Toolbox™ User's Guide*



#### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

#### Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

**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) |

# Contents

## 1 Getting Started


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

**About Data Servers and Data Service Providers** . . . . . 1-3
Supported Data Service Providers . . . . . 1-3
Data Server Connection Requirements . . . . . 1-4


## 2 Communicate with Financial Data Servers


**Communicate with Data Providers** . . . . . 2-2

**Connect to the Bloomberg Communications Server** . . . 2-3

**Connection Object Properties** . . . . . 2-5
Retrieve Connection Properties . . . . . 2-5
Retrieve Data on a Security . . . . . 2-6

**Disconnect from Data Providers** . . . . . 2-7


## 3 Example: Retrieve Bloomberg Data


**About This Example** . . . . . 3-2

**Retrieve Field Data** . . . . . 3-3

**Retrieve Time-Series Data** . . . . . **3-4**

**Retrieve Historical Data** . . . . . **3-5**

## 4 Datafeed Toolbox Graphical User Interface

**Introduction** . . . . . **4-2**

**Retrieve Data with the Datafeed Dialog Box** . . . . . **4-3**
Connecting to Data Servers . . . . . **4-4**
Retrieving Data . . . . . **4-5**
Setting Overrides . . . . . **4-6**

**Obtain Ticker Symbol with Datafeed Securities Lookup** . . . . . **4-9**

## 5 Functions — Alphabetical List

1

# Getting Started


- "Datafeed Toolbox Product Description" on page 1-2
- "About Data Servers and Data Service Providers" on page 1-3

# 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 develop realistic models 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.

## Key Features

- Current, intraday, historical, and real-time market data access
- Customizable data access by security lists, time periods, and other fields
- Intraday tick data retrieval as a time series
- Real-time security data access
- Bloomberg®, Thomson Reuters™, and other data server provider support
- Haver Analytics® and Federal Reserve Economic Data (FRED®) economic data support

# About Data Servers and Data Service Providers

**In this section...**

"Supported Data Service Providers" on page 1-3

"Data Server Connection Requirements" on page 1-4

## Supported Data Service Providers

This toolbox supports connections to financial data servers that the following corporations provide:

- Bloomberg L.P. (`http://www.bloomberg.com`)

  **Note** Only Bloomberg Desktop API is supported.

- eSignal® (`http://www.esignal.com`)
- FactSet® Research Systems, Inc. (`http://www.factset.com`)
- Federal Reserve Economic Data (FRED) (`http://research.stlouisfed.org/fred2/`)
- Haver Analytics ( `http://www.haver.com`)
- Interactive Data™ (`http://www.interactivedata-prd.com/`)
- IQFEED®(`http://www.iqfeed.net/`)
- Kx Systems®, Inc. (`http://www.kx.com`)
- SIX Financial Information (`http://www.six-financial-information.com`)
- Thomson Reuters (`http://www.thomsonreuters.com/`)
- Yahoo!® (`http://finance.yahoo.com`)

See the MathWorks® Web site for the system requirements for connecting to these data servers.

## Data Server Connection Requirements

To connect to some of these data servers, additional requirements apply.

### Additional Software Requirements

The following data service providers require you to install proprietary software on your PC:

- Bloomberg

> **Note** You must have a Bloomberg Desktop software license for the host on which the Datafeed Toolbox and MATLAB software are running.

- Interactive Data
- Haver Analytics
- Kx Systems, Inc.
- Reuters®
- IQFEED

You must have a valid license for required client software on your machine. If you do not, the following error message appears when you try to connect to a data server:

```
Invalid MEX-file
```

For details about how to obtain required software, contact your data server sales representative.

### Proxy Information Requirements

The following data service providers may require you to specify a proxy host and proxy port plus a user name and password if the user's site requires proxy authentication:

- FactSet
- FRED

- Thomson Reuters Datastream®
- Thomson Reuters Tick History
- Yahoo!
- IQFEED

For information on how to specify these settings, see "Specify Proxy Server Settings for Connecting to the Internet".

## FactSet Data Service Requirements

You need a license to use FactSet FAST technology. For details, see the FactSet Web site at `http://www.factset.com`.

## Reuters Data Service Requirements

**Configuring Reuters Connections Using the Reuters Configuration Editor.** You must use the Reuters Configuration Editor to configure your connections as follows:

1. Open the Reuters Market Data System configuration editor by typing the following command:

   ```
   rmdsconfig
   ```

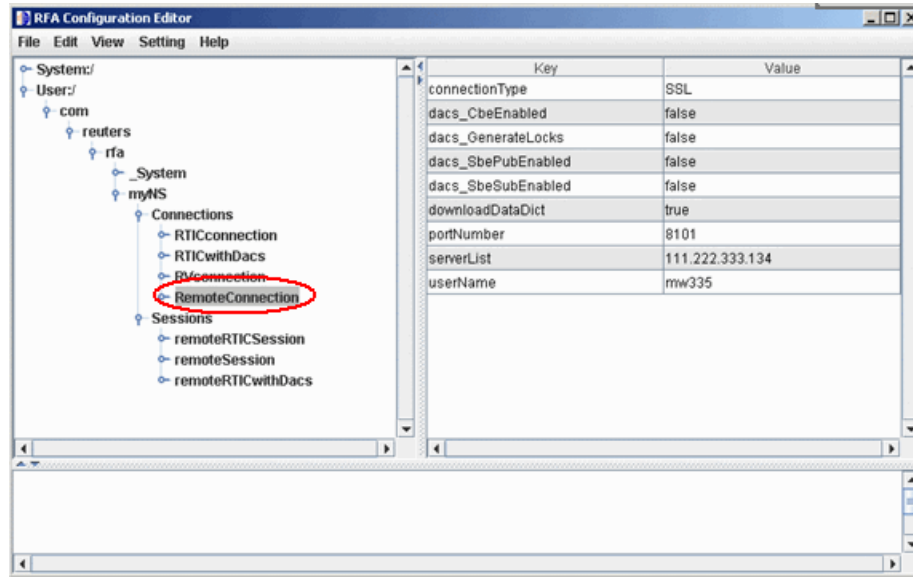
2. Load the sample configuration file.

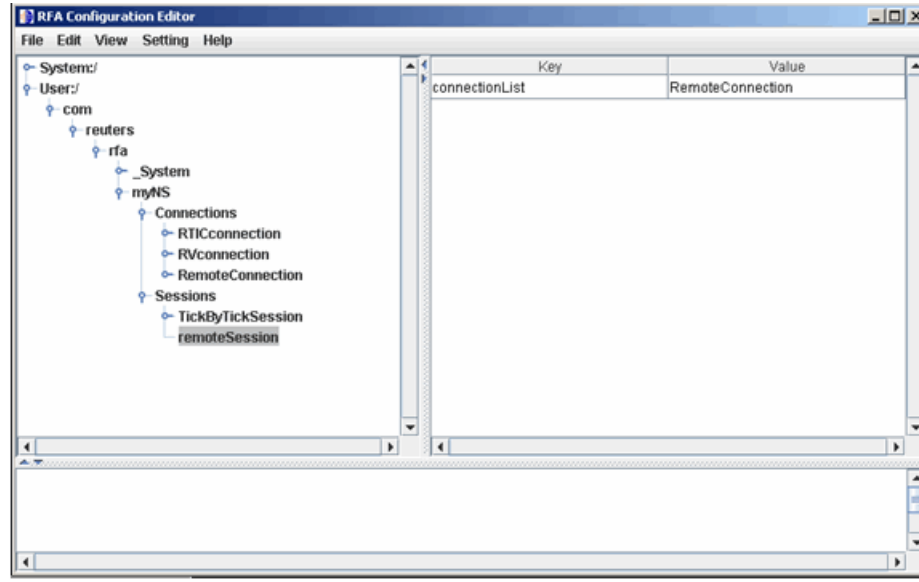
   a. Click **File > Import > File**.

   b. Select the file *matlabroot*`\toolbox\datafeed\datafeed\sampleconfig.xml`.

3. Modify `sampleconfig.xml` based on the site-specific settings that you obtain from Reuters.

4. Define a namespace, a connection, and a session associated with the connection. The following example adds the session `remoteSession` with the namespace `MyNS` to the connection list for the connection `remoteConnection`.




**5** If you are not DACS-enabled, disable DACS.

**a** Add the following to your connection configuration:

```
dacs_CbeEnabled=false
dacs_SbePubEnabled=false
dacs_SbeSubEnabled=false
```

**b** If you are running an SSL connection, add the following to your connection configuration:

```
dacs_GenerateLocks=false
```

For details, see the `reuters` function reference page.

**Troubleshooting Issues with Reuters Configuration Editor.** These errors occur when you attempt to use the Reuters Configuration Editor to configure connections on a machine on which an XML Parser is not installed.

```
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 problem, download an XML parser file, and then include a path to this file in your `CLASSPATH` environment variable.

The following example shows how to set your `CLASSPATH` environment variable to include the XML parser file `C:\xerces.jar` (downloaded from `http://xerces.apache.org/xerces-j/index.html`):

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

### Thomson Reuters Data Service Requirements

You need the following to connect to Thomson Reuters data servers:

- A license for Thomson Reuters Datastream DataWorks®.
- To connect to the Thomson Reuters Datastream API from the Web, you need a user name, password, and URL provided by Thomson Reuters.

For details, see the Thomson Reuters Web site at `http://www.thomsonreuters.com`.

2

# Communicate with Financial Data Servers


- "Communicate with Data Providers" on page 2-2
- "Connect to the Bloomberg Communications Server" on page 2-3
- "Connection Object Properties" on page 2-5
- "Disconnect from Data Providers" on page 2-7

# Communicate with Data Providers

This section uses Bloomberg as an example of how to retrieve data with the Datafeed Toolbox. To establish a connection to the Bloomberg communications server, use `blp`. To retrieve connection properties, use `get`. To terminate a connection, use `close`.

You can communicate with other supported data providers using a similar set of toolbox functions. The table below lists functions used to connect to different data providers.

| Data Provider | Toolbox Function |
|---|---|
| Bloomberg | `blp` |
| eSignal | `esig` |
| FactSet | `factset` or `fds` |
| FRED | `fred` |
| Haver Analytics | `haver` |
| Interactive Data | `idc` |
| IQFEED | `iqf` |
| Kx Systems, Inc. | `kx` |
| SIX Financial Information | `tlkrs` |
| Thomson Reuters | `datastream` or `reuters` |
| Yahoo! | `yahoo` |

To retrieve connection properties, use `get`. To terminate a connection, use `close`.

# Connect to the Bloomberg Communications Server

This example shows how to use the `blp` function to connect to the Bloomberg communications server.

1. If you have not used the `blp` function before, you will need to add the file `blpapi3.jar` to the MATLAB Java® class path. Use the `javaaddpath` function or edit your `javaclasspath.txt` file.

   ---

   **Note** With the Bloomberg V3 release, there is a Java archive file from Bloomberg that you need to install for `blp` and other commands work correctly. If you already have `blpapi3.jar` downloaded from Bloomberg, you can find it in your Bloomberg folders at: `..\blp\api\APIv3\JavaAPI\lib\blpapi3.jar` or `..\blp\api\APIv3\JavaAPI\v3.3.1.0\lib\blpapi3.jar`.

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

   a. In your Bloomberg terminal, type `WAPI {GO}` to display the **Desktop/Server API** screen.

   b. Select **SDK Download Center** and then click **Desktop v3.x API**.

   c. Once you have `blpapi3.jar` on your system, add it to the MATLAB Java javaclasspath using `javaaddpath`. This is must be done for every session of MATLAB. To avoid repeating this at every session, you can add `javaaddpath` to your `startup.m` file or you can add the full path for `blpapi3.jar` to your `javaclasspath.txt` file. For details, see "Bringing Java Classes into MATLAB Workspace".

   ---

2. Enter the following command:

   ```
   c = blp
   ```

   You are now connected to the Bloomberg Communications Server. Your output appears as follows:

   ```
   c =
   ```

```
  session: [1x1 com.bloomberglp.blpapi.Session]
ipaddress: 'localhost'
     port: 8194.00
```

# Connection Object Properties

| In this section... |
| --- |
| "Retrieve Connection Properties" on page 2-5 |
| "Retrieve Data on a Security" on page 2-6 |

The syntax for the Bloomberg V3 connection object constructor is:

```
c = blp;
```

## Retrieve Connection Properties

To retrieve the properties of a connection object, use the `get` function. This function returns different values depending upon which data server you are using.

```
get(c)

c =

      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: 'localhost'
         port: 8194.00
```

You can get the values of the individual properties by using the property names:

```
get(c,{'port','session'})

ans =

       port: 8194.00
    session: [1x1 com.bloomberglp.blpapi.Session]
```

For example, return just the connection handle with the `ipaddress` argument:

```
ip = get(c,{'ipaddress'})
ip =
    localhost
```

**Note** A single property is not returned as a structure.

## Retrieve Data on a Security

Establish a connection, `b`, to a Bloomberg communications server:

```
c = blp;
```

Use `timeseries` to return data on a security:

```
d = timeseries(c,'IBM US Equity',floor(now));
```

To return data on a particular field for a range of dates, use `history`:

```
data = history(c,'IBM US Equity','Last_Price','07/15/2009','08/02/2009')
```

# Disconnect from Data Providers

To close a data provider connection and disconnect, use `close` with the format:

```
close(c)
```

You must have previously created the connection object with one of the connection functions.

3

# Example: Retrieve Bloomberg Data


- "About This Example" on page 3-2
- "Retrieve Field Data" on page 3-3
- "Retrieve Time-Series Data" on page 3-4
- "Retrieve Historical Data" on page 3-5

# About This Example

The following example illustrates the use of the `blp` functions to retrieve data from a Bloomberg communications server.

---

**Note** If you have not used the `blp` function before you will need to add the file `blpapi3.jar` to the MATLAB Java class path. Use the `javaaddpath` function or edit your `javaclasspath.txt` file. For details, see "Bringing Java Classes into MATLAB Workspace".

---

# Retrieve Field Data

`getdata` obtains Bloomberg field data. The entire set of field data provides statistics for all possible securities, but it does not apply universally to any one security.

To obtain data for specific fields of a given security, use the `getdata` function with the following syntax:

```
d = getdata(Connect, Security, Fields)
```

For example, use the Bloomberg connection object `c` to retrieve the values of the fields `Open` and `Last_Price`:

```
c = blp
d = getdata(c,'IBM US Equity', {'Open';'Last_Price'})
d =
    Open: 126.2500
    Last_Price: 125.1250
```

## Retrieve Time-Series Data

`timeseries` returns price and volume data for a particular security on a specified date. Use the following command to return time-series data for a given security and a specific date:

```
data = timeseries(Connection, Security, Date)
```

`Date` can be a MATLAB date string or serial date number.

To obtain time-series data for the current day, use the alternate form of the function:

```
data = timeseries(Connection, Security, floor(now))
```

To obtain time-series data for `IBM` using an existing connection `c`, enter the following:

```
c = blp
data = timeseries(c, 'IBM US Equity', floor(now));
```

# Retrieve Historical Data

Use `history` to obtain historical data for a specific security.

To obtain historical data for a specified field of a particular security, run:

```
d = history(Connect,Security,Field,FromDate,ToDate)
```

`history` returns data for the date range from `FromDate` through `ToDate`.

For example, to obtain the closing price for `IBM` for the dates July 15, 2009 through August 2, 2009 using the connection `c`, enter:

```
c = blp
data = history(c, 'IBM US Equity', 'Last_Price',...
'07/15/2009', '08/02/2009');
```

4

# Datafeed Toolbox Graphical User Interface


- "Introduction" on page 4-2
- "Retrieve Data with the Datafeed Dialog Box" on page 4-3
- "Obtain Ticker Symbol with Datafeed Securities Lookup" on page 4-9

# Introduction

You can use the Datafeed Toolbox Graphical User Interface (GUI) to connect to and retrieve information from some supported data service providers.

This GUI consists of two dialog boxes:

- The Datafeed dialog box
- The Securities Lookup dialog box

# Retrieve Data with the Datafeed Dialog Box

The Datafeed dialog box establishes the connection with the data server and manages data retrieval. Use this dialog box to connect to and retrieve data from the following service providers:

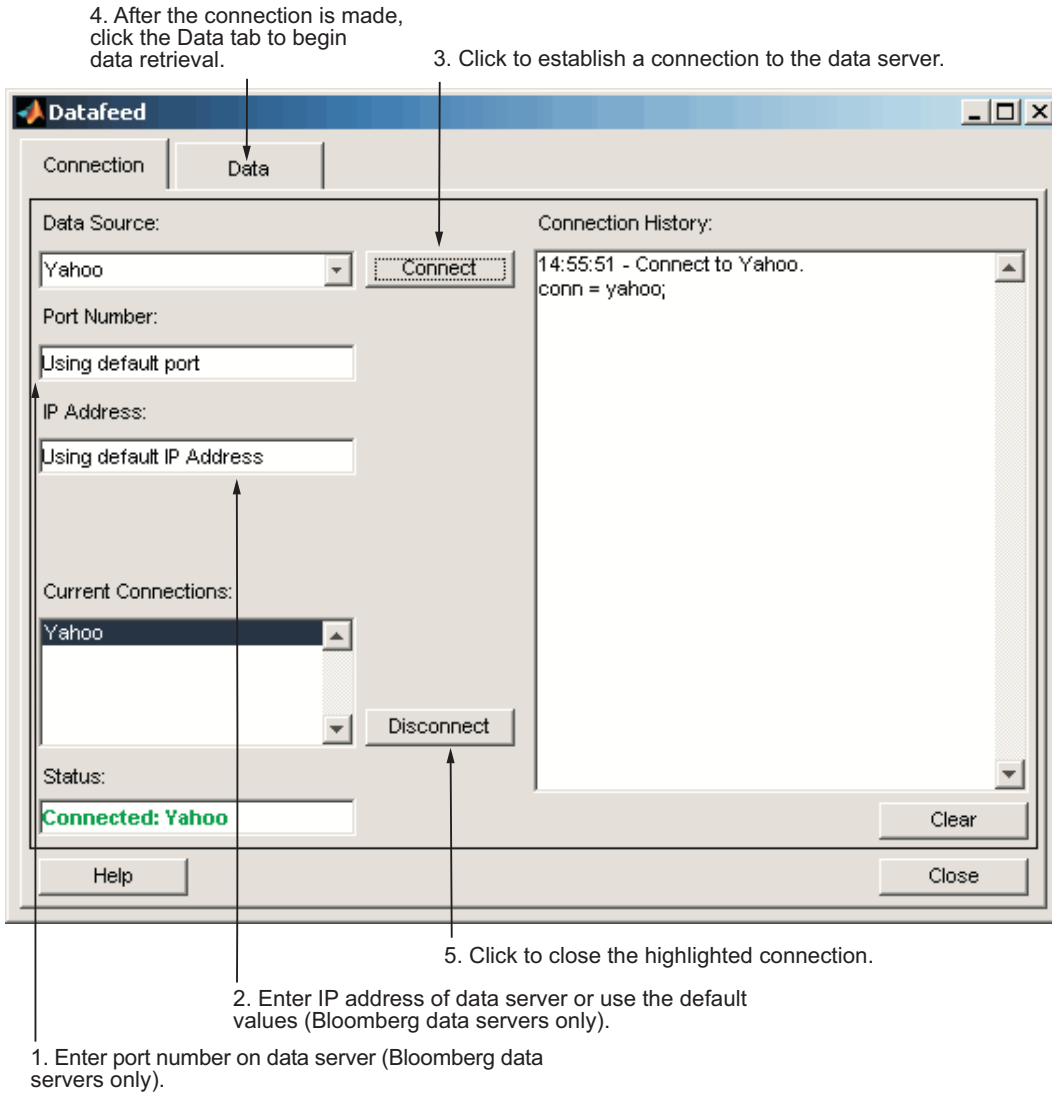
- Bloomberg
- Interactive Data
- Yahoo!

To display this dialog box, enter the `dftool` command in the MATLAB Command Window.

The Datafeed dialog box consists of two tabs:

- The **Connection** tab establishes communication with a data server. For details, see "Connecting to Data Servers" on page 4-4.
- The **Data** tab specifies the data request. For details, see "Retrieving Data" on page 4-5.
- You can also set overrides for the data you retrieve. For details, see "Setting Overrides" on page 4-6.

The following figure summarizes how to connect to data servers and retrieve data using the Datafeed dialog box.




**The Datafeed Dialog Box**

## Connecting to Data Servers

**1** Click the **Connect** button to establish a connection.

2 When the `Connected` message appears in the **Status** field, click the **Data** tab to begin the process of retrieving data from the data server. For details, see "Retrieving Data" on page 4-5.

3 Click the **Disconnect** button to terminate the session highlighted in the **Current Connections** box.

For Bloomberg data servers, you must also specify the port number and IP address of the server:

1 Enter the port number on the data server in the **Port Number** field.

2 Enter the IP address of the data server in the **IP Address** field.

3 To establish a connection to the Bloomberg data server, follow steps 1 through 3 above.

---

**Tip** You can also connect to the Bloomberg data server by selecting the **Connect** button and accepting the default values.

---

## Retrieving Data

The **Data** tab allows you to retrieve data from the data server as follows:

1 Enter the security symbol in the **Enter Security** field.

2 Indicate the type of data to retrieve in the **Data Selection** field.

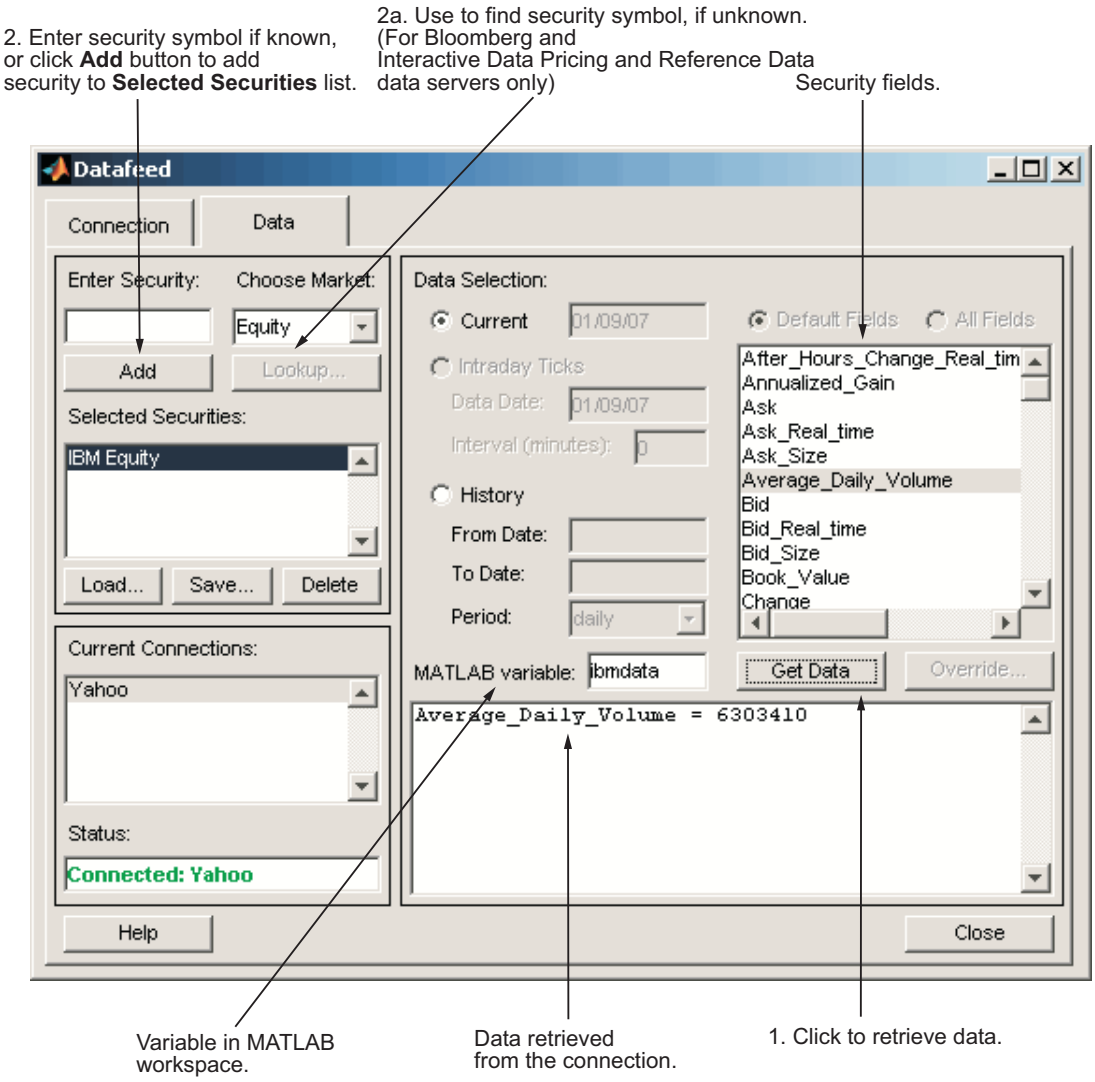
3 Specify whether you want the default set of data, or the full set:

- Select the **Default fields** button for the default set of data.
- Select the **All fields** button for the full set of data.

4 Click the **Get Data** button to retrieve the data from the data server.

5 (Optional) Click the **Override** button if you want to set overrides on the data you request from the data server. For details, see "Setting Overrides" on page 4-6.

The following figure summarizes these steps.



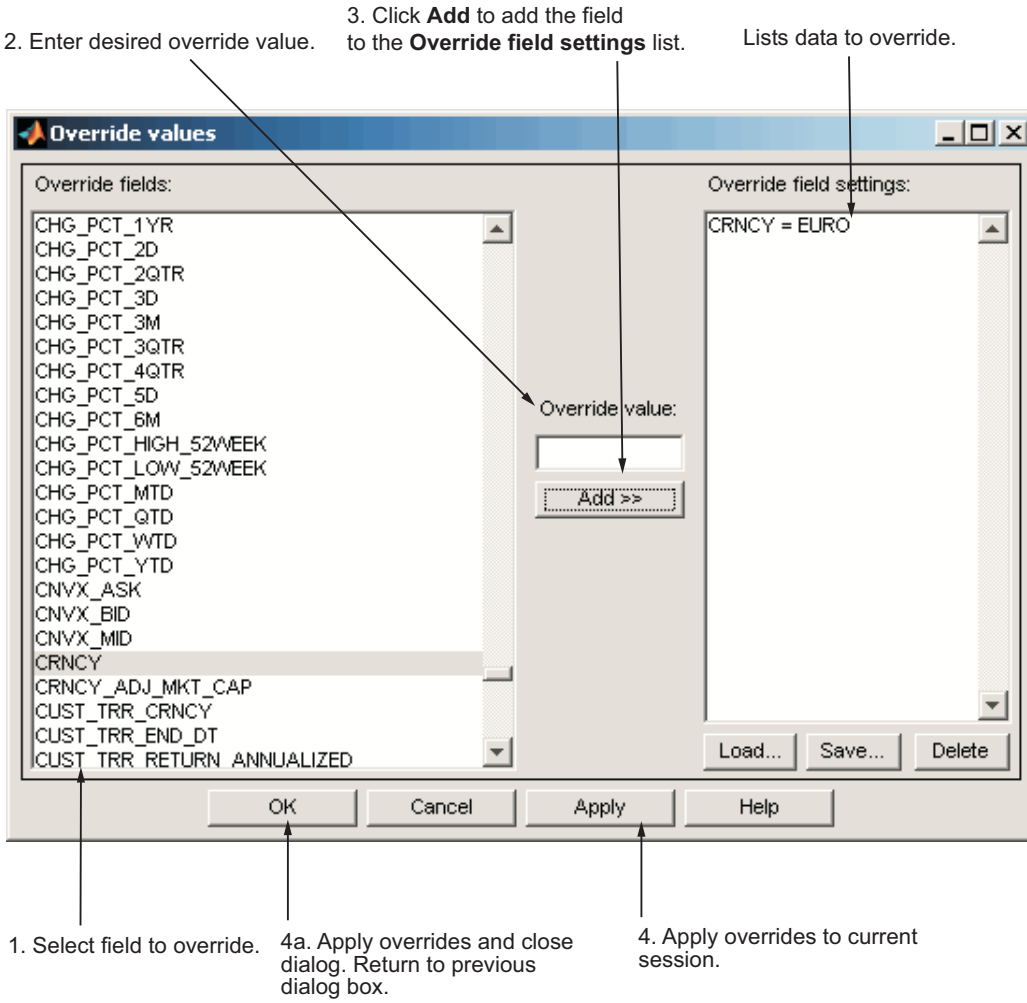

## Setting Overrides

To set overrides on retrieved data:

1 Click the **Override** button. The Override values dialog box opens.

2. Select the field to override from the **Override fields** selection list.
3. Enter the desired override value in the **Override value** field.
4. Click **Add** to add the field to override to the **Override field settings** list.
5. Click **Apply** to apply overrides to the current session and keep the Override values dialog box open, or click **OK** to apply the overrides and close the dialog box.

The following figure summarizes these steps.


2. Enter desired override value.
3. Click **Add** to add the field to the **Override field settings** list.
Lists data to override.
Override values
Override fields:
CHG_PCT_1YR
CHG_PCT_2D
CHG_PCT_2QTR
CHG_PCT_3D
CHG_PCT_3M
CHG_PCT_3QTR
CHG_PCT_4QTR
CHG_PCT_5D
CHG_PCT_6M
CHG_PCT_HIGH_52WEEK
CHG_PCT_LOW_52WEEK
CHG_PCT_MTD
CHG_PCT_QTD
CHG_PCT_WTD
CHG_PCT_YTD
CNVX_ASK
CNVX_BID
CNVX_MID
CRNCY
CRNCY_ADJ_MKT_CAP
CUST_TRR_CRNCY
CUST_TRR_END_DT
CUST_TRR_RETURN_ANNUALIZED
Override value:
Add >>
Override field settings:
CRNCY = EURO
Load...
Save...
Delete
OK
Cancel
Apply
Help
1. Select field to override.
4a. Apply overrides and close dialog. Return to previous dialog box.
4. Apply overrides to current session.

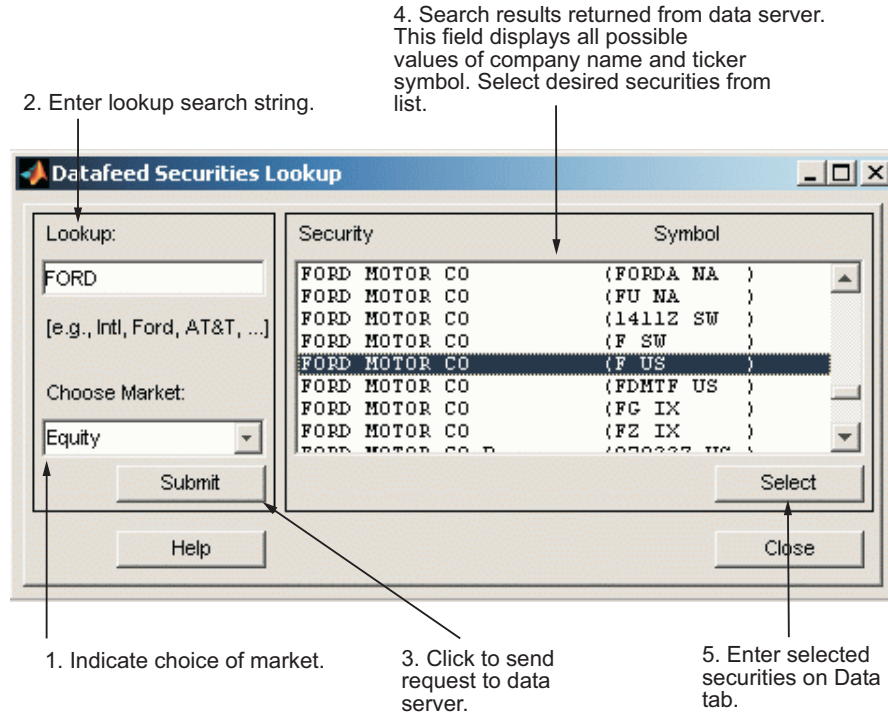
# Obtain Ticker Symbol with Datafeed Securities Lookup

When requesting data from Bloomberg or Interactive Data servers, you can use the Datafeed Securities Lookup dialog box to obtain the ticker symbol for a given security if you know only part of the security name.

1. Click the **Lookup** button on the Datafeed dialog box **Data** tab. The Securities Lookup dialog box opens.
2. Specify your choice of market in the **Choose Market** field.
3. Enter the known part of the security name in the **Lookup** field.
4. Click **Submit**. All possible values of the company name and ticker symbol corresponding to the security name you specified display in the **Security** and **Symbol** list.
5. Select one or more securities from the list, and then click **Select**.

   The selected securities are added to the **Selected Securities** list on the **Data** tab.

The following figure summarizes these steps.


4. Search results returned from data server. This field displays all possible values of company name and ticker symbol. Select desired securities from list.
2. Enter lookup search string.
Datafeed Securities Lookup
Lookup:
FORD
[e.g., Intl, Ford, AT&T, ...]
Choose Market:
Equity
Submit
Help
Security
Symbol
FORD MOTOR CO (FORDA NA )
FORD MOTOR CO (FU NA )
FORD MOTOR CO (1411Z SW )
FORD MOTOR CO (F SW )
FORD MOTOR CO (F US )
FORD MOTOR CO (FDMTF US )
FORD MOTOR CO (FG IX )
FORD MOTOR CO (FZ IX )
Select
Close
1. Indicate choice of market.
3. Click to send request to data server.
5. Enter selected securities on Data tab.

5

# Functions — Alphabetical List

# dftool

## Purpose

Datafeed dialog box

## Syntax

```
dftool
```

## Description

The Datafeed dialog box establishes the connection with the data server and manages data retrieval. Use this dialog box to connect to and retrieve data from the following service providers:

- Bloomberg
- Interactive Data
- Yahoo!

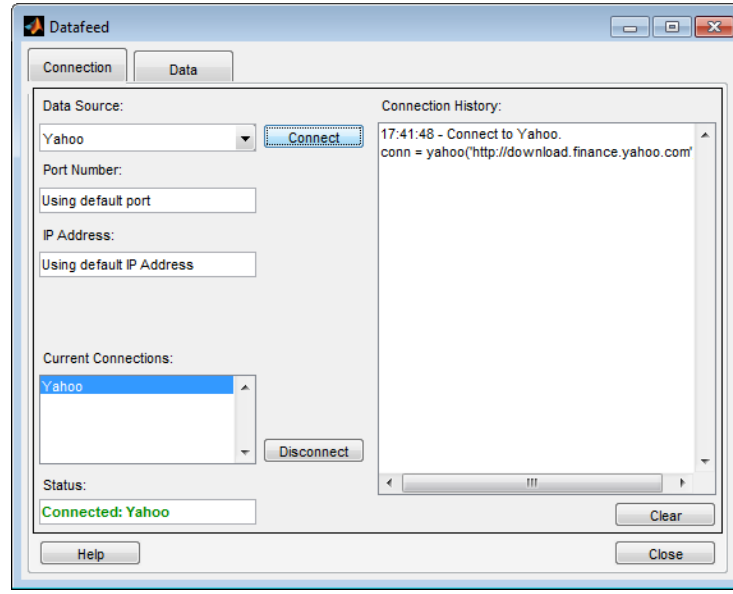
To display this dialog box, enter the `dftool` command in the MATLAB Command Window.

The Datafeed dialog box consists of two tabs:

- The **Connection** tab establishes communication with a data server. For details, see "Connecting to Data Servers" on page 4-4.
- The **Data** tab specifies the data request. For details, see "Retrieving Data" on page 4-5.
- You can also set overrides for the data you retrieve. For details, see "Setting Overrides" on page 4-6.

## Examples

```
dftool
```




## How To

- "Retrieve Data with the Datafeed Dialog Box" on page 4-3

# bloomberg

## Purpose

Connect to Bloomberg data servers

`bloomberg` is not recommended. Use `blp` instead.

## Syntax

```
c = bloomberg
```

## Description

`c = bloomberg` establishes a connection, `c`, to a Bloomberg data server. It uses port number `8194` and the default internet address provided when you installed the Bloomberg software on your machine.

## Examples

Establish a connection, `c`, to a Bloomberg data server:

```
c = bloomberg
```

## See Also

`close` | `fetch` | `get` | `isconnection`

## Purpose

Close connections to Bloomberg data servers

`bloomberg` is not recommended. Use `blp` instead.

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Bloomberg connection object created with the `bloomberg` function. |

## Description

`close(Connect)` closes the connection to the Bloomberg data server.

## Examples

Establish a Bloomberg connection `c`:

```
c = bloomberg
```

Close this connection:

```
close(c)
```

## See Also

`bloomberg`

# fetch

## Purpose

Request data from Bloomberg data servers

`fetch` is not recommended. Use the `blp` functions `getdata`, `history`, `realtime`, or `timeseries` instead.

## Syntax

```
data = fetch(Connect, 'Security')
data = fetch(Connect, 'Security', 'HEADER', 'Flag', 'Ident')
data = fetch(Connect, 'Security', 'GETDATA', 'Fields', 'Override',
   'Values', 'Ident')
data = fetch(Connect, 'Security', 'TIMESERIES', 'Date', 'Minutes',
   'TickField')
data = fetch(Connect, 'Security', 'HISTORY', 'Fields', 'FromDate',
   'ToDate', 'Period', 'Currency', 'Ident')
ticker = fetch(Connect, 'SearchString', 'LOOKUP', 'Market')
data = fetch(Connect, 'Security', 'REALTIME', 'Fields',
   'MATLABProg')
data = fetch(Connect, 'Security', 'STOP')
```

## Description

For a given security, `fetch` returns header (default), current, time-series, real-time, and historical data via a connection to a Bloomberg data server.

`data = fetch(Connect, 'Security')` fills the header fields with data from the most recent date with a bid, ask, or trade.

`data = fetch(Connect, 'Security', 'HEADER', 'Flag', 'Ident')` returns data for the most recent date of each individual field for the specified security type identifiers, based upon the value of `Flag`.

- If `'Flag'` is `'DEFAULT'`, `fetch` fills the header fields with data from the most recent date with a bid, ask, or trade. Alternatively, you could use the command `data = fetch(Connect,'Security')`.
- If `'Flag'` is `'TODAY'`, `fetch` returns the header field data with data from today only.
- If `'Flag'` is `'ENHANCED'`, `fetch` returns the header field data for the most recent date of each individual field. In this case, for example, the bid and ask group fields could come from different dates.

`data = fetch(Connect, 'Security', 'GETDATA', 'Fields', 'Override', 'Values', 'Ident')` returns the current market data for the specified fields of the indicated security. You can further specify the data with the optional `Override`, `Values` and `Ident` arguments.

---

**Note** If a call to the fetch function with the `GETDATA` argument encounters an invalid security in a list of securities to retrieve, it returns `NaN` data for the invalid security's fields.

---

`data = fetch(Connect, 'Security', 'TIMESERIES', 'Date', 'Minutes', 'TickField')` returns the tick data for a single security for the specified date. You can further specify data with the optional `Minutes` and `TickField` arguments. If there is no data found in the specified range, which must be no more than 50 days, `fetch` returns an empty matrix.

You can specify `TickField` as a string or numeric value. For example, `TickField = 'Trade'` or `TickField = 1` returns data for ticks of type `Trade`. The function `dftool('ticktypes')` returns the list of intraday tick fields. `fetch` returns intraday tick data requested with an interval with the following columns:

- Time
- Open
- High
- Low
- Value of last tick
- Volume total value of ticks
- Total value of ticks for the time range
- Number of ticks

The `fetch` function returns columns 7 and 8 only if they make sense for the requested field.

For today's tick data, enter the command:

```
data = fetch(Connect,'Security', 'TIMESERIES', now)
```

For today's trade time series aggregated into 5-minute intervals, enter:

```
data = fetch(Connect,'Security','TIMESERIES', ...
now, 5, 'Trade')
```

`data = fetch(Connect, 'Security', 'HISTORY', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency', 'Ident')` returns historical data for the specified field for the date range `FromDate` through `ToDate`. You can set the time period with the optional `Period` argument to return a more specific data set. You can further specify returned data by appending the `Currency` or `Ident` argument.

---

**Note** If a call to the `fetch` function with the `HISTORY` argument encounters an invalid security in a list of securities to retrieve, it returns no data for any securities in the list.

---

`ticker = fetch(Connect, 'SearchString', 'LOOKUP', 'Market')` uses `SearchString` to find the ticker symbol for a security trading in a designated market. The output `ticker` is a column vector of possible ticker values.

---

**Note** If you supply `Ident` without a period or currency, enter `[]` for the missing values.

---

`data = fetch(Connect, 'Security', 'REALTIME', 'Fields', 'MATLABProg')` subscribes to a given security or list of securities, requesting the indicated fields, and runs any specified MATLAB function. See `pricevol`, `showtrades`, or `stockticker` for information on the data returned by asynchronous Bloomberg events.

`data = fetch(Connect, 'Security', 'STOP')` unsubscribes the list of securities from processing Bloomberg real-time events.

## Arguments

`Connect` Bloomberg connection object created with the `bloomberg` function.

`'Security'` A MATLAB string containing the name of a security, or a cell array of strings containing a list of securities, specified in a format recognizable by the Bloomberg server. You can substitute a `CUSIP number` for a security name as needed. You can only call a single security when using the `TIMESERIES` flag as well.

**Note** This argument is case sensitive.

`'Flag'` A MATLAB string indicating the dates for which to retrieve data. Possible values are:

- `DEFAULT`: Data from most recent bid, ask, or trade. If you do not specify a `Flag` value, `fetch` uses the default value of `'DEFAULT'`.
- `TODAY`: Today's data only.
- `ENHANCED`: Data from most recent date of each individual field.

`'Currency'` (Optional) Currency in which the `fetch` function returns historical data. A list of valid currencies appears in the file `@bloomberg/bbfields.mat`. Default = `[]`.

`'Ident'` (Optional) Security type identifier. A list of valid currencies appears in the file `@bloomberg/bbfields.mat`. Default = `[]`.

`'Fields'` A MATLAB string or cell array of strings specifying specific fields for which you request data. A list of valid currencies appears in the file `@bloomberg/bbfields.mat`. Default = `[]`.

`'Override'` (Optional) String or cell array of strings containing override field list. Default = `[]`.

| | |
|---|---|
| `'Values'` | (Optional) String or cell array of strings containing override field values. |
| `'Date'` | Date string, serial date number, or cell array of dates that specifies dates for the time-series data. Specify `now` to retrieve today's time-series data. |
| `'Minutes'` | (Optional) Numeric value for tick interval in minutes. You cannot specify a fractional value for `'Minutes'`. The smallest value you can specify is 1. |
| `'TickField'` | (Optional) You can specify a string or numeric value for this field. For example, `TickField = 'Trade'` or `TickField = 1` return data for ticks of type `Trade`. Use the command `dftool('ticktypes')` to return the list of intraday tick fields. |
| `'FromDate'` | Beginning date for historical data. |

**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 historical data. |
| `'Period'` | (Optional) Period of the data. A MATLAB three-part string with the format:<br>`'Frequency Days Data'`<br>`Frequency` Values:<br>• `d`: Daily (default)<br>• `w`: Weekly<br>• `m`: Monthly<br>• `q`: Quarterly<br>• `s`: Semiannually<br>• `y`: Yearly<br>`Days` Values: |

- `o`: Omit all days for which there is no data (default)
- `i`: Include all trading days
- `a`: Include all calendar days

`Data` Values:

- `b`: Report missing data using Bloomberg (default)
- `s`: Show missing data as last found value
- `n`: Report missing data as `NaN`

  For example, `'dan'` returns daily data for all calendar days, reporting missing values as `NaN`. If a value is unspecified, `fetch` returns a default value.

**Note** If you do not specify a value for `Period`, `fetch` uses default values.

`'Currency'` (Optional) Currency type. The file `@bloomberg/bbfields.mat` lists supported currencies.

`'Market'` A MATLAB string indicating the market in which a particular security trades. Possible values are:

- `Comdty`: (Commodities)
- `Corp`: (Corporate bonds)
- `Equity`: (Equities)
- `Govt`: (Government bonds)
- `Index`: (Indexes)
- `M-Mkt`: (Money Market securities)
- `Mtge`: Mortgage-backed securities)

- `Muni`: (Municipal bonds)
- `Pfd`: (Preferred stocks)

`'MATLABProg'` A string that is the name of any valid MATLAB program.

## Examples

### Retrieving Header Data

Retrieve header data for a U.S. equity with ticker `ABC`:

```
D = fetch(c,'ABC US Equity')
```

### Retrieving Opening and Closing Prices

Retrieve the opening and closing prices:

```
D = fetch(c,'ABC US Equity','GETDATA',...
{'Last_Price';'Open'})
```

### Retrieving Override Fields

Retrieve the requested fields, given override fields and values:

```
D = fetch(c, '3358ABCD4 Corp', 'GETDATA',...
{'YLD_YTM_ASK', 'ASK', 'OAS_SPREAD_ASK', 'OAS_VOL_ASK'},...
{'PX_ASK', 'OAS_VOL_ASK'}, {'99.125000', '14.000000'})
```

### Retrieving Time-Series Data

Retrieve today's time series:

```
D = fetch(c, 'ABC US Equity', 'TIMESERIES', now)
```

### Retrieving Time-Series Data, Aggregated into Time Intervals

Retrieve today's trade time series for the given security, aggregated into five-minute intervals:

```
D = fetch(c, 'ABC US Equity', 'TIMESERIES', now, 5,'Trade')
```

### Retrieving Time-Series Default Closing Price

Retrieve the closing price for the given dates, using the default period of the data:

```
D = fetch(c, 'ABC US Equity', 'HISTORY', 'Last_Price', ...
'8/01/99', '8/10/99')
```

### Retrieving Monthly Closing Price

Retrieve the monthly closing price for the specified dates:

```
D = fetch(c, 'ABC US Equity', 'HISTORY', 'Last_Price', ...
'8/01/99', '9/30/00', 'm')
```

## See Also

`bloomberg` | `close` | `get` | `isconnection`

## Purpose

Retrieve Bloomberg connection object properties

`get` is not recommended. Use the `blp` function `get` instead.

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Bloomberg connection object created with the `bloomberg` function. |
| `PropertyName` | (Optional) A MATLAB string or cell array of strings containing property names. Property names are:<br>• `'Connection'`<br>• `'IPAddress'`<br>• `'Port'`<br>• `'Socket'`<br>• `'Version'` |

## Description

`value = get(Connect, 'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the Bloomberg connection object.

`value = get(Connect)` returns the value for all properties.

## Examples

Establish a connection, `c`, to a Bloomberg data server:

```
c = bloomberg
```

Retrieve this connection's properties:

```
p = get(c, {'Port', 'IPAddress'})
p =
     port: 8194
     ipaddress: 111.222.33.444
```

## See Also

bloomberg | close | fetch | isconnection

# isconnection

## Purpose

Verify whether connections to Bloomberg data servers are valid

`bloomberg` is not recommended. Use `blp` instead.

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Bloomberg connection object created with the `bloomberg` function. |

## Description

`x = isconnection(Connect)` returns `x = 1` if the connection to the Bloomberg data server is valid, and `x = 0` otherwise.

## Examples

Establish a connection, `c`, to a Bloomberg data server:

```
c = bloomberg
```

Verify that `c` is a valid connection:

```
x = isconnection(c)
x = 1
```

## See Also

`bloomberg` | `close` | `fetch` | `get`

## Purpose

Verify if valid Bloomberg field

`bloomberg` is not recommended. Use `blp` instead.

## Syntax

```
x = isfield(c,f)
```

## Description

`x = isfield(c,f)` returns true if specified field, `f`, is a valid Bloomberg field and false otherwise. `f` can be a cell array of strings. `c` is the Bloomberg connection handle.

## Examples

```
x = isfield(c,{'LAST_PRICE','VOLUME','OPEN','HIGH'})
```

returns

```
 x =           1     1     1     1
```

## See Also

`bloomberg.close` | `bloomberg.fetch` | `bloomberg.get` | `bloomberg.isconnection`

# lookup

## Purpose

Bloomberg security search

`bloomberg` is not recommended. Use `blp` instead.

## Syntax

```
d = lookup(c,s,market)
```

## Description

`d = lookup(c,s,market)` returns the list of matching securities given the security search string `s` and market `m`. The `lookup` function uses the Bloomberg ActiveX® interface.

## Examples

The command

```
D = LOOKUP(c,'Intl Bus Mac','Equity')
```

returns the securities along with their ticker symbols matching the search string `'Intl Bus Mac'` for the `Equity` market. Valid market types are:

- `Comdty`: (Commodities)
- `Corp`: (Corporate bonds)
- `Equity`: (Equities)
- `Govt`: (Government bonds)
- `Index`: (Indexes)
- `M-Mkt`: (Money Market securities)
- `Mtge`: Mortgage-backed securities)
- `Muni`: (Municipal bonds)
- `Pfd`: (Preferred stocks)

## See Also

`fetch`

## Purpose

Bloomberg V3 communications server connection

## Syntax

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

## Description

`c = blp` connects to the local Bloomberg V3 communications server. You need a Bloomberg Desktop software license for the host on which the Datafeed Toolbox and MATLAB software are running.

---

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

---

`c = blp(p,ip,timeout)` connects to the local Bloomberg communications server.

## Input Arguments

### p - Port number

[ ] (default) | scalar

Port number, specified as a scalar to identify the port number of the local machine where Bloomberg is running.

**Data Types**
`double`

### ip - IP address

[ ] (default) | string

IP address, specified as a string to identify the local machine where Bloomberg is running.

**Data Types**
`char`

### timeout - Timeout value

scalar

Timeout value, specified as a scalar to denote the time in milliseconds the local machine attempts to connect before timing out if the connection cannot be established.

**Data Types**
`double`

## Output Arguments

### c - Bloomberg V3 connection

connection object

Bloomberg V3 connection, returned as a connection object. The properties of this object are as follows.

| Property | Description |
|---|---|
| `session` | Bloomberg V3 API COM object |
| `ipaddress` | IP address |
| `port` | Port number |
| `timeout` | Number in milliseconds specifying how long MATLAB attempts to connect to a Bloomberg V3 communications server before timing out |

## Examples

### Connect to a Bloomberg Communications Server

Establish a connection `c` to a Bloomberg communications server.

```
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.

### Connect to a Bloomberg Communications Server with a Timeout

Establish a connection using the default port and `'localhost'` as the IP address, with a timeout value of 10,000 milliseconds.

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

```
c =
  blp with properties:
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: 'localhost'
         port: 8194
      timeout: 10000
```

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

## Tips

With the Bloomberg V3 release, there is a Java archive file from Bloomberg that you need to install for `blp` and other Bloomberg commands to work correctly.

If you already have `blpapi3.jar` downloaded from Bloomberg, you can find it in your Bloomberg folders at: `..\blp\api\APIv3\JavaAPI\lib\blpapi3.jar` or `..\blp\api\APIv3\JavaAPI\v3.3.1.0\lib\blpapi3.jar`. If you have `blpapi3.jar`, go to step 3.

If `blpapi3.jar` is not downloaded from Bloomberg, then download it as follows:

1. In your Bloomberg terminal, type `WAPI {GO}` to open the Desktop/Server API screen.
2. Select **SDK Download Center**, and then click **Desktop v3.x API**.

3 Once you have `blpapi3.jar` on your system, add it to the MATLAB Java class path using `javaaddpath`.

   You need to do this for every session of MATLAB. To avoid repeating this at every session, add `javaaddpath` to your `startup.m` file or add the full path for `blpapi3.jar` to your `javaclasspath.txt` file. For details about `javaclasspath.txt`, see "The Java Class Path". For details about editing your `startup.m` file, see "Specifying Startup Options in MATLAB Startup File".

## See Also

`category` | `close` | `fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

## Purpose

Bloomberg V3 field category search

## Syntax

```
d = category(c,f)
```

## Description

`d = category(c,f)` returns category information given a search term `f`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### f - Search term

string

Search term, specified as a string to denote Bloomberg fields.

**Data Types**

`char`

## Output Arguments

### d - Return data

cell array

Return data, returned as an N-by-5 cell array containing categories, field identifiers, field mnemonics, field names, and field data types for each N row in the data set.

## Examples

### Search for the Bloomberg Last Price Field

Create the Bloomberg connection.

```
c = blp;
```

Request the Bloomberg category description of the last price field.

```
d = category(c,'LAST_PRICE');
```

Display the first three rows of Bloomberg category description data in `d`.

```
d(1:3,:)

ans =

    'Analysis'    'OPO54'    'DELTA_LAST'    'Delta Last Trade...'    'Do
    'Analysis'    'OPO51'    'IVOL_LAST'     'Implied Volatili...'    'Do
    'Analysis'    'OPOO6'    'DELTA'         'Delta Best Price'       'Do
```

The columns in `d` contain the following:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

Close the Bloomberg connection.

```
close(c);
```

## See Also

`fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

## Purpose

Close connection to Bloomberg V3

## Syntax

```
close(c)
```

## Description

`close(c)` closes the Bloomberg V3 connection `c`.

## Input Arguments

### `c` - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

## Examples

### Close the Bloomberg Connection

Create the Bloomberg connection object `c` using `blp`.

```
c = blp;
```

Close the Bloomberg connection using the Bloomberg connection object `c`.

```
close(c);
```

## See Also

`blp`

## Purpose

Return equity screening data from Bloomberg V3

## Syntax

```
d = eqs(c,sname)
d = eqs(...,stype)
d = eqs(...,languageid)
d = eqs(...,group)
```

## Description

`d = eqs(c,sname)` returns equity screening data given the Bloomberg V3 session screen name `sname`.

`d = eqs(...,stype)` returns equity screening data using the screen type `stype`. `stype` can be set to `'GLOBAL'` for Bloomberg screen names or `'PRIVATE'` for customized screen names.

`d = eqs(...,languageid)` returns equity screening data using the language identifier `languageid`.

`d = eqs(...,group)` returns equity screening data using the optional group identifier `group`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### sname - Screen name

string

Screen name, specified as a string 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`

### stype - Screen type

'GLOBAL' | 'PRIVATE'

Screen type, specified as one of the two enumerated strings above 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.

**Data Types**
char

### languageid - Language identifier

string

Language identifier, specified as a string to denote the language for the returned data. This argument is optional.

**Data Types**
char

### group - Group identifier

string

Group identifier, specified as a 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

## Output Arguments

### d - Return data

cell array

Return data, returned as a cell array containing Bloomberg equity screening data.

## Examples

### Retrieve Equity Screening Data for a Screen

Create the Bloomberg connection.

```
c = blp;
```

Retrieve equity screening data for the screen called `Frontier Market Stocks with 1 billion USD Market Caps`.

```
d = eqs(c,'Frontier Market Stocks with 1 billion USD Market Caps');
```

Display the first three rows in the returned data `d`.

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

```
ans =

  Columns 1 through 4

    'Cntry'           'Name'                    'Ind Group'    'Market Cap
    'Bahrain'         'ARAB BANKING COR...'     'Banks'        [1166249984
    'South Africa'    'HARMONY GOLD MIN...'     'Mining'       [1239142656

  Columns 5 through 8

    'Price:D-1'    'P/B'     'P/E'     'EPS - 1 Yr Gr LF'
    [     0.38]    [0.30]    [5.18]    [           24.53]
    [     2.89]    [0.40]    [ NaN]    [          -96.84]
```

`d` contains Bloomberg equity screening data for the `Frontier Market Stocks with 1 billion USD Market Caps` screen. The first row contains column headers and the subsequent rows contain the returned data. The columns in `d` contain the following:

- Country name
- Company name
- Industry name

- Market capitalization
- Price
- Price-to-book ratio
- Price-earnings ratio
- Earnings per share

Close the connection.

```
close(c);
```

### Retrieve Equity Screening Data for a Screen Type

Create the Bloomberg connection.

```
c = blp;
```

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
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36

  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 and the subsequent rows contain the returned data. The columns in `d` contain the following:

- 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;
```

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...'    'Pre
  'HON     US'    'HONEYWELL INTL'    [     69451382784.00]    [
  'CMI     US'    'CUMMINS INC'       [     24799526912.00]    [

Columns 6 through 8

  'Gesamtertrag YTD'    'Erl s 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 and the subsequent rows contain the returned data. The columns in `d` contain the following:

- 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;
```

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'
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51]
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36]

  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 and the subsequent rows contain the returned data. The columns in `d` contain the following:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio

- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c);
```

## See Also

`blp` | `getdata` | `tahistory`

# fieldinfo

## Purpose

Bloomberg V3 field information

## Syntax

```
d = fieldinfo(c,f)
```

## Description

`d = fieldinfo(c,f)` returns field information on Bloomberg V3 connection object `c` given a field mnemonic `f`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### f - Field mnemonic

string

Field mnemonic, specified as a string that is used to retrieve Bloomberg field information.

**Data Types**

`char`

## Output Arguments

### d - Return data

cell array

Return data, returned as an N-by-5 cell array containing the field help, field identifier, field mnemonic, field name, and field data type.

## Examples

### Retrieve Information for the Bloomberg Last Price Field

Create a Bloomberg connection.

```
c = blp;
```

Retrieve the Bloomberg field information for the `LAST_PRICE` field.

```
d = fieldinfo(c,'LAST_PRICE');
```

Display the returned Bloomberg information.

```
celldisp(d)

d{1} =

Last price for the security. Field updates in realtime.

Equities:
  Returns the last price provided by the exchange. For securities that
...

d{2} =

RQ005


d{3} =

LAST_PRICE


d{4} =

Last Trade/Last Price


d{5} =

Double
```

The columns in `d` contain the following:

- Field help with the Bloomberg descriptive information
- Field identifier
- Field mnemonic
- Field name

- Field data type

Close the Bloomberg connection.

```
close(c);
```

## See Also

`category` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

## Purpose

Bloomberg V3 field search

## Syntax

```
d = fieldsearch(c,f)
```

## Description

`d = fieldsearch(c,f)` returns field information on Bloomberg V3 connection object `c` given a search string `f`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### f - Search term

string

Search term, specified as a string that is used to retrieve Bloomberg field descriptive data.

**Data Types**

`char`

## Output Arguments

### d - Return data

cell array

Return data, returned as an N-by-5 cell array containing categories, field identifiers, field mnemonics, field names, and field data types for each N row in the data set.

## Examples

### Search for the Bloomberg Last Price Field

Create a Bloomberg connection.

```
c = blp;
```

Return data for the search string `LAST_PRICE`.

```
d = fieldsearch(c,'LAST_PRICE');
```

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

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

```
ans =

    'Market Activity/...'    'PR005'    'PX_LAST'              'Last Pric
    'Market Activity/...'    'RQ005'    'LAST_PRICE'           'Last Trad
    'Market Activity/...'    'RQ134'    'LAST_ALL_SESSIONS'    'Last Pric
```

The columns in `d` contain the following:

- Category
- Field identifier
- Field mnemonic
- Field name
- Field data type

Close the Bloomberg connection.

```
close(c);
```

## See Also

`category` | `fieldinfo` | `getdata` | `history` | `realtime` | `timeseries`

## Purpose

Get Bloomberg V3 connection properties

## 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.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### properties - Property names

string | cell array

Property names, specified as a string or cell array of strings containing Bloomberg connection property names. The property names are `session`, `ipaddress`, `port`, and `timeout`.

**Data Types**

`char | cell`

## Output Arguments

### v - Bloomberg connection properties

scalar | string | object | structure

Bloomberg connection properties, returned as a scalar if the port number or timeout is requested, a string if the IP address is requested, an object if the Bloomberg session is requested, or a structure if all properties are requested.

## Examples

### Retrieve Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

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;
```

Retrieve the port number from the Bloomberg connection object by specifying `'port'` as a string.

```
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;
```

Create a cell array `properties` with strings `'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);
```

## See Also

`getdata` | `history` | `realtime` | `timeseries` | `blp` | `close`

# getdata

## Purpose

Current Bloomberg V3 data

## Syntax

```
[d,sec] = getdata(c,s,f)
[...] = getdata(...,o,ov)
[...] = getdata(...,Name,Value)
```

## Description

`[d,sec] = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`.

`[...] = getdata(...,o,ov)` returns the data using the override fields `o` with corresponding override values `ov`.

`[...] = getdata(...,Name,Value)` returns the data using `Name`,`Value` pair arguments for additional Bloomberg request settings.

## 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.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### s - Security list

string | cell array

Security list, specified as a string for one security or a cell 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`

### f - Bloomberg data fields

string | cell array

Bloomberg data fields, specified as a Bloomberg specific string for one data field or a cell array of Bloomberg specific strings for multiple data fields. For details about the strings 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`

### o - Bloomberg override field

string | cell array

Bloomberg override field, specified as a Bloomberg specific string for one data field or a cell array of Bloomberg specific strings for multiple data fields. For details about the strings 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`

### ov - Bloomberg override field value

string | cell array

Bloomberg override field value, specified as a string for one Bloomberg override field or a cell array of strings for multiple Bloomberg override fields. Use this field value to filter the Bloomberg data result set from the `getdata` function.

**Example:** `'20100101'`

**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 single 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 a Boolean where `true` adds a name and value for the EID date to the return data.

**Data Types**
`logical`

### 'returnFormattedValue' - Return format

true | false

Return format, specified as a Boolean where `true` forces all data to return as a data type string.

**Data Types**
`logical`

### 'useUTCTime' - Date time format

true | false

Date time format, specified as a Boolean where `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 Boolean where `true` returns the latest data up to the delay period specified by the exchange for the security.

**Data Types**
`logical`

## Output Arguments

### d - Bloomberg return data
structure

Bloomberg return data, returned as a structure with the Bloomberg data. For details about the returned data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

### sec - Security list
cell array

Security list, returned as a cell array of strings 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

## Examples

### Return the Closing and Open Price of the Given Security

Create the Bloomberg connection.

```
c = blp;
```

Request closing 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 closing and open prices. Also, `getdata` returns the security in `SEC`.

Close the connection.

```
close(c);
```

### Return the Requested Fields Given Override Fields and Values

Create the Bloomberg connection.

```
c = blp;
```

Request data for Bloomberg fields `YLD_YTM_ASK`, `ASK`, and `OAS_SPREAD_ASK` when the Bloomberg field `OAS_VOL_ASK` equals `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);
```

### Return a Request for IBM® Using its CUSIP Number

Create the Bloomberg connection.

```
c = blp;
```

Request the closing 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 closing price.

Close the connection.

```
close(c);
```

### Return the Closing Price for the Security with a Pricing Source BGN

Create the Bloomberg connection.

```
c = blp;
```

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 closing price.

Close the connection.

```
close(c);
```

### Return the Constituent Weights Using a Date Override

Create the Bloomberg connection.

```
c = blp;
```

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);
```

## See Also

blp | history | realtime | timeseries

# history

## Purpose

Bloomberg V3 historical data

## Syntax

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

## Description

`[d,sec] = 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` and `sec`, are sorted to match the input order of `s`.

`[...] = history(...,period)` returns the historical data for the fields `f` and the dates `fromdate` through `todate` with a specific periodicity `period`.

`[...] = history(...,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`.

`[...] = history(...,Name,Value)` returns the historical data for the security list `s` using `Name,Value` pair arguments for additional Bloomberg request settings.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### s - Security list

string | cell array

Security list, specified as a string for one security or a cell 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`

### f - Bloomberg data fields
string | cell array of strings

Bloomberg data fields, specified as a Bloomberg specific string for one data field or a cell array of Bloomberg specific strings for multiple data fields. For details about the strings 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`

### period - Periodicity
`daily | weekly | monthly | quarterly | semi_annually | ...`

Periodicity, specified as a cell array of enumerated strings to denote the period of the data to return. For example, when `period` is set to `{'daily','calendar'}`, the `history` function returns daily data for all calendar days reporting missing data as `NaN`s. When `period` is set to`{'actual'}`, the `history` function returns the data using the default periodicity and default calendar reporting missing data as `NaN`s. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. The default calendar is actual trading days. The possible values of `period` are as follows.

| Value | Time Period |
|---|---|
| `daily` | Daily |
| `weekly` | Weekly |

| Value | Time Period |
|---|---|
| `monthly` | Monthly |
| `quarterly` | Quarterly |
| `semi_annually` | Semiannually |
| `yearly` | Yearly |
| `actual` | Anchor date specification |
| `calendar` | Anchor date specification |
| `fiscal` | Anchor date specification |
| `non_trading_weekdays` | Non-trading weekdays |
| `all_calendar_days` | Return all calendar days |
| `active_days_only` | Active trading days only |
| `previous_value` | Fill missing values with previous values |
| `nil_value` | Fill missing values with a `NaN` |

**Data Types**
`char | cell`

### currency - Currency
string

Currency, specified as a string 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`

### fromdate - Beginning date
scalar | vector | matrix | string | cell array

Beginning date for the historical data, specified as a double scalar, double vector, double matrix, string, or cell array of strings. You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

**Data Types**
`double | char | cell`

#### todate - End date
scalar | vector | matrix | string | cell array

End date for the historical data, specified as a double scalar, double vector, double matrix, string, or cell array of strings. You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

**Data Types**
`double | 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 single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `'adjustmentNormal',true`

#### 'adjustmentNormal' - Historical normal pricing adjustment
true | false

Historical normal pricing adjustment, specified as 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, and 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 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, and 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 a Boolean to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, and 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 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 return data

matrix

Bloomberg return data, returned as a matrix with the Bloomberg data. The first column of the matrix is the numeric representation of the date. The remaining columns contain the requested data fields. For details about the return data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

### sec - Security list

cell array

Security list, returned as a cell array of strings 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`

## Examples

### Retrieve the Daily Closing Price for a Date Range

Create the Bloomberg connection.

```
c = blp;
```

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);
```

### Retrieve the Monthly Closing Price for a Date Range

Create the Bloomberg connection.

```
c = blp;
```

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);
```

### Retrieve the Monthly Closing Price for a Date Range Using U.S. Currency

Create the Bloomberg connection.

```
c = blp;
```

Get the monthly closing price from August 1, 2010 through December 10, 2010 for the IBM security in U.S. 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);
```

### Retrieve the Monthly Closing Price for a Date Range Using U.S. Currency with a Specified Period

Create the Bloomberg connection.

```
c = blp;
```

Get the monthly closing price from August 1, 2010 through August 10, 2010 for the IBM security in U.S. currency `USD`. The period values `daily`, `actual`, and `all_calendar_days` specify returning actual daily 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','8/10/2010',{'daily','actual',...
                  'all_calendar_days','nil_value'},'USD')

d =

     734351.00           NaN
```

```
    734352.00       123.55
    734353.00       123.18
    734354.00       124.03
    734355.00       124.56
    734356.00       123.58
    734357.00          NaN
    734358.00          NaN
    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);
```

### Retrieve the Weekly Closing Price for a Date Range Using U.S. Currency

Create the Bloomberg connection.

```
c = blp;
```

Get the monthly closing price from November 1, 2010 through December 23, 2010 for the IBM security in U.S. currency `USD`. Note that the anchor date depends on the date December 23, 2010 in this case. Because this date is a Thursday, each previous value is reported for the Thursday of the week in question.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '11/01/2010','12/23/2010',{'weekly'},'USD')
```

```
d =

     734446.00        139.39
     734453.00        138.71
     734460.00        137.69
     734467.00        139.07
     734474.00        138.47
     734481.00        137.63
     734488.00        137.87
     734495.00        139.15


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);
```

### Retrieve the Closing Price for a Date Range Using U.S. Currency with the Default Period

Create the Bloomberg connection.

```
c = blp;
```

Get the closing price from August 1, 2010 through September 10, 2010 for the IBM security in U.S. currency `USD` with the default period of the data set using `[]`. The default period of a security depends on the security itself.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','9/10/2010',[],'USD')
```

```
d =

     734352.00        123.55
     734353.00        123.18
     734354.00        124.03
     ...

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);
```

### Retrieve the Daily Closing Price for a Date Range Using U.S. Currency with Name-Value Pairs

Create the Bloomberg connection.

```
c = blp;
```

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);
```

### Retrieve the Daily Closing Price Using a CUSIP Number with a Pricing Source

Create the Bloomberg connection.

```
c = blp;
```

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 = 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
     ...
```

d contains the numeric representation for the date in the first column and the closing price in the second column.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve the Closing Price for a Date Range Using an International Date Format

Create the Bloomberg connection.

```
c = blp;
```

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);
```

### Retrieve the Median Estimated Earnings Per Share Using an Override Value

Create the Bloomberg connection.

```
c = blp;
```

Retrieve the median estimated earnings per share for AkzoNobel® from October 1, 2010 through October 30, 2010. When specifying Bloomberg override fields, use the string `'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'},'CapChg',true)

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);
```

## Definitions

### Anchor Date

The *anchor date* is the date to which all other reported dates are related. For `blp.history`, for periodicities other than daily, `ToDate` is the

anchor date. For example, if you set the period to weekly and the `ToDate` is a Thursday, every reported data point would also be a Thursday, or the nearest prior business day to Thursday. Similarly, if you set the period to monthly and the `ToDate` is the 20th of a month, each reported data point would be for the 20th of each month in the date range.

## See Also

`blp` | `realtime` | `timeseries` | `getdata`

# isconnection

## Purpose

Validate Bloomberg V3 connection

## Syntax

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

## Description

`v = isconnection(c)` returns `true` if `c` is a valid Bloomberg V3 connection and `false` otherwise.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

## Output Arguments

### v - Valid Bloomberg connection

true | false

Valid Bloomberg connection, specified as a logical true, 1, or a logical false, 0.

## Examples

### Validate the Bloomberg Connection

Create the Bloomberg connection.

```
c = blp;
```

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);
```

## See Also

`blp` | `close` | `getdata`

# lookup

## Purpose

Bloomberg V3 lookup to find information about securities

## 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.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### q - Keyword query

string

Keyword query, specified as a string containing one or more strings for each item for which information is requested. For example, the keyword query string can be a security, a curve type, or a filter ticker.

**Data Types**
`char`

### reqtype - Request type

`'instrumentListRequest' | 'curveListRequest' | 'govtListRequest'`

Request type, specified as the above enumerated strings 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.

**Data Types**
`char`

## 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 single 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

scalar

Number of rows in the result data, specified as a 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

string

Bloomberg yellow key filter, specified as a unique string to denote the particular yellow key for government securities, corporate bonds, equities, and commodities, for example.

**Data Types**
`char`

### 'languageOverride' - Language override

string

Language override, specified as a unique string to denote a translation language for the result data.

**Data Types**
`char`

### 'countryCode' - Country code

string

Country code, specified as a string to denote the country for the result data.

**Data Types**

`char`

### 'currencyCode' - Currency code

string

Currency code, specified as a string to denote the currency for the result data.

**Data Types**

`char`

### 'curveID' - Bloomberg short-form identifier for curve

string

Bloomberg short-form identifier for a curve, specified as a string.

**Data Types**

`char`

### 'type' - Bloomberg market sector type

string

Bloomberg market sector type corresponding to the Bloomberg yellow keys, specified as a string.

**Data Types**

`char`

### 'subtype' - Bloomberg market sector subtype

string

Bloomberg market sector subtype, specified as a string to further delineate the market sector type.

**Data Types**
`char`

### 'partialMatch' - Partial match on ticker
`true | false`

Partial match on ticker, specified as `true` or `false`. When set to `true`, you can filter securities by setting `q` to a query string such as `'T*'`. When set to `false`, the securities are unfiltered.

**Data Types**
`logical`

## Output Arguments

### l - Lookup information
structure

Lookup information, returned as a structure containing set properties depending on the request type. For a list of properties and their description, see the following tables.

The properties for the `'instrumentListRequest'` request type are as follows.

| Property | Description |
|---|---|
| `security` | Security name |
| `description` | Security long name |

The properties for the `'curveListRequest'` request type are as follows.

| Property | Description |
|---|---|
| `curve` | Bloomberg curve name |
| `description` | Bloomberg description |
| `country` | Country code |
| `currency` | Currency code |

| Property | Description |
|---|---|
| curveid | Bloomberg short-form identifier for the curve |
| type | Bloomberg market sector type |
| subtype | Bloomberg market sector subtype |
| publisher | Bloomberg is the publisher |
| bbgid | Bloomberg identifier |

The properties for the 'govtListRequest' request type are as follows.

| Property | Description |
|---|---|
| parseky | Bloomberg security identifier (ticker or CUSIP, for example), price source, and source key (Bloomberg yellow key). |
| name | Government security name |
| ticker | Government security ticker |

## Examples

### Look Up a Security

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.

Connect to Bloomberg.

```
c = blp;
```

Retrieve the instrument data for an IBM corporate bond with a maximum of 20 rows of data.

```
insts = lookup(c,'IBM','instrumentListRequest','maxResults',20,...
               'yellowKeyFilter','YK_FILTER_CORP',...
               'languageOverride','LANG_OVERRIDE_NONE')
```

```
insts =

         security: {20x1 cell}
      description: {20x1 cell}
```

The Security Lookup returns the security names and descriptions.

Display the `IBM` corporate bond names.

```
insts.security
```

```
ans =
  'IBM<corp>'
  'IBM GB USD SR 10Y<corp>'
  'IBM GB USD SR 3Y<corp>'
  'IBM GB USD SR 30Y<corp>'
  'IBM GB USD SR 5Y<corp>'
  'IBM CDS USD SR 5Y<corp>'
  'BL037645<corp>'
  'IBM CDS USD SR 3Y<corp>'
  'IBM CDS USD SR 1Y<corp>'
  'BL106695<corp>'
  'IBM CDS USD SR 10Y<corp>'
  'IBM CDS USD SR 4Y<corp>'
  'IBM CDS USD SR 6Y<corp>'
  'IBM CDS USD SR 30Y<corp>'
  'IBM CDS USD SR 7Y<corp>'
  'IBM CDS USD SR 15Y<corp>'
  'BF106693<corp>'
  'IBMTR<corp>'
  'IBM CDS USD SR 2Y<corp>'
  'IBM CDS USD SR 0M<corp>'
```

Display the `IBM` corporate bond descriptions.

```
insts.description
```

```
ans =
```

```
'International Business Machines Corp (Multiple Matches)'
'International Business Machines Corp Generic Benchmark 10Y Corporate
'International Business Machines Corp Generic Benchmark 3Y Corporate'
'International Business Machines Corp Generic Benchmark 30Y Corporate
'International Business Machines Corp Generic Benchmark 5Y Corporate'
'International Business Machines Corp'
'IBM Loan USD REV 11/10/2017'
'International Business Machines Corp'
'International Business Machines Corp'
'IBM Loan JPY TL 06/30/2017'
'International Business Machines Corp'
'International Business Machines Corp'
'International Business Machines Corp'
'International Business Machines Corp'
'International Business Machines Corp'
'International Business Machines Corp'
'IBM Loan JPY DEAL 06/30/2017'
'IBM Corp-Backed Interest Rate Putable Underlying Trust 2006-2'
'International Business Machines Corp'
'International Business Machines Corp'
```

Close the Bloomberg connection.

```
close(c);
```

### Look Up a 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;
```

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 a 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;
```

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);
```

## See Also

`blp` | `getdata` | `history` | `realtime` | `timeseries`

# realtime

## Purpose

Bloomberg V3 real-time data retrieval

## Syntax

```
d = realtime(c,s,f)
[subs,t] = realtime(...,eventhandler)
```

## Description

`d = realtime(c,s,f)` returns the data for the given connection `c`, security list `s`, and requested fields `f`.

`[subs,t] = realtime(...,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`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### s - Security list

string | cell array

Security list, specified as a string for one security or a cell 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`

### f - Bloomberg data fields

string | cell array

Bloomberg data fields, specified as a Bloomberg specific string for one data field or a cell array of Bloomberg specific strings for multiple data fields. For details about the strings 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`

### eventhandler - Event handler

string

Event handler, specified as a string denoting 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`

## Output Arguments

### d - Bloomberg return data

structure

Bloomberg return data, returned as a structure with the Bloomberg data. For details about the returned 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`.

## Examples

### Retrieve Data for One Security

Retrieve a snapshot of data for one security only.

Create the Bloomberg connection.

```
c = blp;
```

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;
```

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;
```

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

`v3stockticker` requires the input argument `f` of `realtime` 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 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 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;
```

Retrieve volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for the IBM security using the event handler `v3showtrades`.

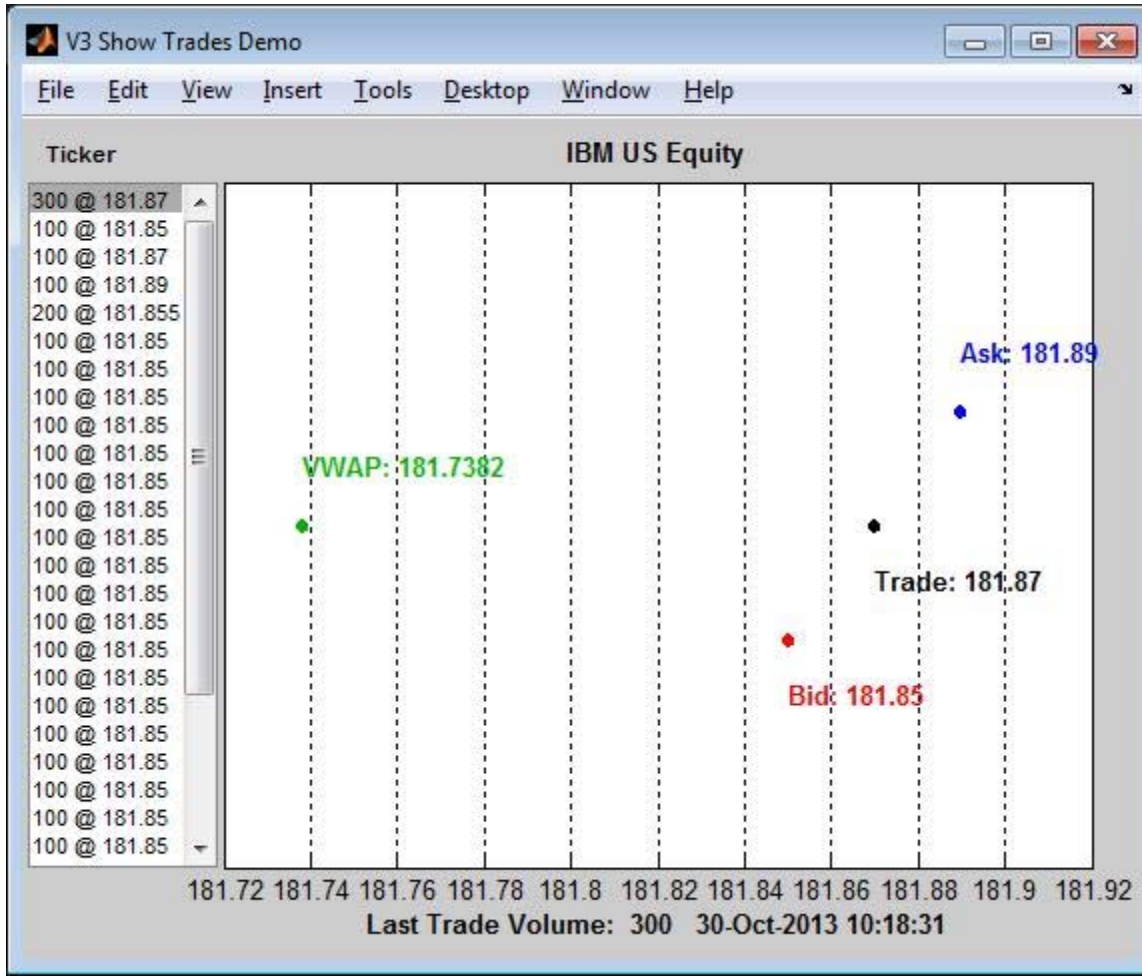
`v3showtrades` requires the input argument `f` of `realtime` to be any combination of: `Last_Trade`, `Bid`, `Ask`, `Volume`, and `VWAP`.

```
[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;
```

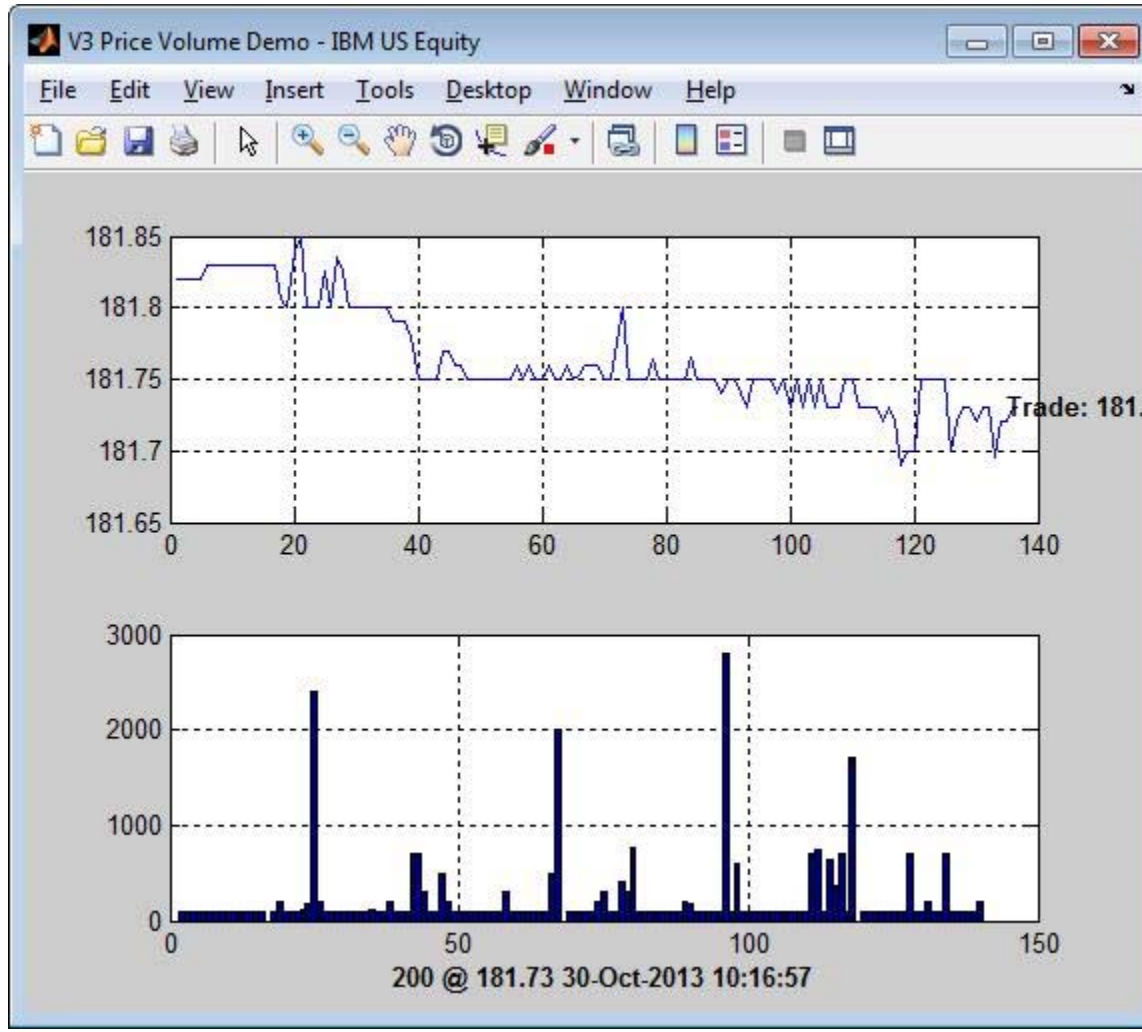
Retrieve last price and volume data for the IBM security using event handler `v3pricevol`.

`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);
```

## See Also

`blp` | `history` | `timeseries` | `stop` | `close`

## Purpose

Unsubscribe real-time requests for Bloomberg 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.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### 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

string | cell array

Security list, specified as a string for one security or a cell 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`

## Examples

### Stop Real-Time Requests

Unsubscribe to real-time data for one security.

Create the Bloomberg connection.

```
c = blp;
```

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;
```

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);
```

## See Also

`blp` | `getdata` | `history` | `realtime` | `timeseries`

# tahistory

## Purpose

Return historical technical analysis from Bloomberg 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.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### s - Security

string

Security, specified as a string for a single Bloomberg security.

**Data Types**

`char`

### startdate - Start date

scalar | string

Start date, specified as a scalar or string to denote the start date of the date range for the returned tick data.

**Example:** `floor(now-1)`

**Data Types**

`double | char`

### enddate - End date

scalar | string

End date, specified as a scalar or string to denote the end date of the date range for the returned tick data.

**Example:** `floor(now)`

**Data Types**

`double | char`

### period - Periodicity

`daily | weekly | monthly | quarterly | semi_annually | ...`

Periodicity, specified as a cell array of enumerated strings to denote the period of the data to return. For example, when `period` is set to `{'daily','calendar'}`, the `history` function returns daily data for all calendar days reporting missing data as `NaN`s. When `period` is set to `{'actual'}` the `history` function returns the data using the default periodicity and default calendar reporting missing data as `NaN`s. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. The default calendar is actual trading days. The possible values of `period` are as follows.

| Value | Time Period |
|---|---|
| `daily` | Daily |
| `weekly` | Weekly |
| `monthly` | Monthly |
| `quarterly` | Quarterly |
| `semi_annually` | Semiannually |
| `yearly` | Yearly |
| `actual` | Anchor date specification |
| `calendar` | Anchor date specification |

| Value | Time Period |
|---|---|
| fiscal | Anchor date specification |
| non_trading_weekdays | Non-trading weekdays |
| all_calendar_days | Return all calendar days |
| active_days_only | Active trading days only |
| previous_value | Fill missing values with previous values |
| nil_value | Fill missing values with a NaN |

**Data Types**
char | cell

### study - Study type
string

Study type, specified as a string to denote the study to use for historical analysis.

**Data Types**
char

## 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 single 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

scalar

Period, specified as a 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

string

High price, specified as a string. For details about the period, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**Data Types**

`char`

### 'priceSourceLow' - Low price

string

Low price, specified as a string. For details about the period, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**Data Types**

`char`

### 'priceSourceClose' - Closing price

string

Closing price, specified as a string. For details about the period, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

**Data Types**

`char`

## Output Arguments

### d - Technical analysis return data

structure

Technical analysis return data, returned as a structure. For details about the possible returned data, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

## Examples

### Request the Bloomberg Directional Movement Indicator (DMI) Study for a 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;
```

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 the Bloomberg Directional Movement Indicator (DMI) Study for a Security with a Pricing Source

Run a technical analysis to return the DMI study for a security with a pricing source.

Create the Bloomberg connection.

```
c = blp;
```

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);
```

## Definitions

### Anchor Date

The *anchor date* is the date to which all other reported dates are related. For `blp.history`, for periodicities other than daily, `ToDate` is the anchor date. For example, if you set the period to weekly and the `ToDate` is a Thursday, every reported data point would also be a Thursday, or the nearest prior business day to Thursday. Similarly, if you set the period to monthly and the `ToDate` is the 20th of a month, each reported data point would be for the 20th of each month in the date range.

## See Also

`blp` | `getdata` | `history` | `realtime` | `timeseries`

# timeseries

## Purpose

Bloomberg V3 intraday tick data

## Syntax

```
d = timeseries(c,s,date)
d = timeseries(...,interval,field)
d = timeseries(c,s,date,[],field,options,values)

d = timeseries(c,s,{startdate,enddate})
d = timeseries(...,interval,field)
d = timeseries(c,s,{startdate,enddate},[],field)
d = timeseries(...,options,values)
```

## Description

`d = timeseries(c,s,date)` retrieves raw tick data `d` for the security `s` and connection object `c` for a specific date `date`.

`d = timeseries(...,interval,field)` retrieves raw tick data `d` for the security `s` and a specific date `date` aggregated into intervals of `interval` for field `field`.

`d = timeseries(c,s,date,[],field,options,values)` retrieves raw tick data `d` for a specific date `date` without an aggregation interval for field `field` with the specified options `options` and corresponding values `values`.

`d = timeseries(c,s,{startdate,enddate})` retrieves raw tick data `d` for security `s` where `startdate` is the starting date and `enddate` is the ending date of the date range.

`d = timeseries(...,interval,field)` retrieves raw tick data `d` for a specific date range aggregated into intervals of `interval` for field `field`.

`d = timeseries(c,s,{startdate,enddate},[],field)` retrieves raw tick data `d` for a specific date range without an aggregation interval for field `field`.

`d = timeseries(...,options,values)` retrieves raw tick data `d` for a specific date range without an aggregation interval for a specific field with specified options `options` and corresponding values `values`.

## Input Arguments

### c - Bloomberg connection

connection object

Bloomberg connection, specified as a connection object created using `blp`.

### s - Security

string

Security, specified as a string for a single Bloomberg security.

**Data Types**
`char`

### date - Date

scalar | string

Date, specified as a scalar or string to denote the specific date for the returned tick data.

**Example:** `floor(now)`

**Data Types**
`double | char`

### interval - Time interval

scalar

Time interval, specified as a scalar to denote the number of minutes between ticks for the returned tick data.

**Data Types**
`double`

### field - Bloomberg field

string

Bloomberg field, specified as a string that defines the tick data to return. Valid values are:

- 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

**Data Types**
`char`

### options - Bloomberg API options
cell array

Bloomberg API options, specified as a cell array of strings. The valid strings are `includeConditionCodes`, `includeExchangeCodes`, and `includeBrokerCodes`.

**Data Types**
`cell`

### values - Bloomberg API values
cell array

Bloomberg API values, specified as a cell array of strings. The valid values are `true` and `false`.

**Data Types**
`cell`

### startdate - Start date
scalar | string

Start date, specified as a scalar or string to denote the start date of the date range for the returned tick data.

**Example:** `floor(now-1)`

**Data Types**
`double | char`

### enddate - End date

scalar | string

End date, specified as a scalar or string to denote the end date of the date range for the returned tick data.

**Example:** `floor(now)`

**Data Types**
`double | char`

## Output Arguments

### d - Bloomberg tick data

cell array | matrix

Bloomberg tick data, returned as a cell array for requests without a specified time interval or a matrix for requests with a specified time interval.

## Examples

### Retrieve Time-Series Tick Data for a Specific Date

Create the Bloomberg connection.

```
c = blp;
```

Retrieve today's trade tick series for the IBM security.

```
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]
    ...
```

`d` contains the tick type in the first column, the numeric representation of the date and time in the second column, the tick value in the third column, and the tick size in the fourth column. Here, the first row shows that 100 IBM shares sold for $181.69 today.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve Time-Series Tick Data for a Specific Date Using a Security with a Pricing Source

Create the Bloomberg connection.

```
c = blp;
```

Retrieve today's trade tick series for the Microsoft security with pricing source `ETPX`.

```
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]
    ...
```

`d` contains the tick type in the first column, the numeric representation of the date and time in the second column, the tick value in the third column, and the tick size in the fourth column. Here, the first row shows that 100 Microsoft shares are sold for $35.53 today.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve Time-Series Tick Data for a Specific Date Using a Time Interval with a Specific Field

Create the Bloomberg connection.

```
c = blp;
```

Retrieve today's trade tick series for the IBM security aggregated into 5-minute intervals.

```
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
   735537.40        181.90        181.97        181.57        181.65
   735537.40        181.73        182.18        181.58        182.07
     ...

  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
- Last price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

Here, the first row of data shows that on today's date the open price is $181.69, the high price is $181.99, the low price is $180.10, the last price is $181.84, the volume is 252,322, the number of ticks is 861, and

the total tick value in the bar is $45,815,588. The next row shows tick data for 5 minutes later.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve Time-Series Tick Data for a Specific Date with a Specific Field and an Option and Value

Create the Bloomberg connection.

```
c = blp;
```

Retrieve today's trade tick series for the `F US Equity` security without specifying the aggregation parameter. Additionally, 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 Time-Series Tick Data Using a Date Range

Create the Bloomberg connection.

```
c = blp;
```

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]
   ...
```

`d` contains the tick type in the first column, the numeric representation of the date and time in the second column, the tick value in the third column, and the tick size in the fourth column. 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);
```

### Retrieve Time-Series Tick Data Using a Date Range with an Interval and a Specific Field

Create the Bloomberg connection.

```
c = blp;
```

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
   735487.40        187.03        187.13        186.65        186.78
   735487.40        186.78        186.78        186.40        186.47
     ...

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
- Last price
- Volume of ticks
- Number of ticks
- Total tick value in the bar

Here, the first row of data shows that on today's date the open price is $187.20, the high price is $187.60, the low price is $187.02, the last price is $187.08, the volume is 207,683, the number of ticks is 560, and

the total tick value in the bar is $38,902,968. The next row shows tick data for 5 minutes later.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve Time-Series Tick Data Using a Date Range with Numerous Fields

Create the Bloomberg connection.

```
c = blp;
```

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]
    ...
```

`d` contains the tick type in the first column, the numeric representation of the date and time in the second column, the tick value in the third column, and the tick size in the fourth column. Here, the first row shows that 100 `F US Equity` security shares sold for $16.71 yesterday.

Close the Bloomberg connection.

```
close(c);
```

### Retrieve Time-Series Tick Data Using a Date Range with Options and Values

Create the Bloomberg connection.

```
c = blp;
```

Return the trade tick series for the security `F US Equity` for yesterday with a time interval at noon, without specifying the aggregation parameter. Additionally, 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. If the equity is one of the former, then the

broker buy code would be in the seventh column and the broker sell code would be 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);
```

## 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 "The 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.

## See Also

`blp` | `history` | `realtime` | `close`

# datastream

## Purpose

Establish connections to Thomson Reuters Datastream API

## Syntax

```
Connect = datastream('UserName', 'Password', 'Source', 'URL')
```

## Arguments

| | |
|---|---|
| `'UserName'` | User name. |
| `'Password'` | User password. |
| `'Source'` | To connect to the Thomson Reuters Datastream API, enter `'Datastream'` in this field. |
| `'URL'` | Web URL. |

**Note** Thomson Reuters assigns the values for you to enter for each argument. Enter all arguments as MATLAB strings.

## Description

`Connect = datastream('UserName', 'Password', 'Source', 'URL')` makes a connection to the Thomson Reuters Datastream API, which provides access to Thomson Reuters Datastream software content.

## Examples

Establish a connection to the Thomson Reuters Datastream API:

```
Connect = datastream('User1', 'Pass1', 'Datastream', ...
'http://dataworks.thomson.com/Dataworks/Enterprise/1.0')
```

**Note** If you get an error connecting, verify that your proxy settings are correct in MATLAB by selecting **Preferences > Web** in the MATLAB Toolstrip.

## See Also

`datastream.close` | `datastream.fetch` | `datastream.get` | `datastream.isconnection`

## Purpose

Close connections to Thomson Reuters Datastream data servers

## 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.

## See Also

`datastream`

# fetch

## Purpose

Request data from Thomson Reuters Datastream data servers

## 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 string containing the name of a security, or cell array of strings containing names of multiple securities. This data is in a format recognizable by the Thomson Reuters Datastream data server. |
| `'Fields'` | (Optional) MATLAB string or cell array of strings indicating the data fields for which to retrieve data. |
| `'Date'` | (Optional) MATLAB 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. |
| | **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 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 strings. For example, it returns `Price` data to the MATLAB workspace as a cell array of strings within the structure. There is no way to determine the data type from the Datastream interface.

---

## 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 Datastream connection.

## See Also

`close` | `datastream` | `get` | `isconnection`

## Purpose

Retrieve properties of Thomson Reuters Datastream connection objects

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Thomson Reuters Datastream connection object created with the `datastream` function. |
| `PropertyName` | (Optional) A MATLAB string or cell array of strings 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.

## See Also

`close` | `datastream` | `fetch` | `isconnection`

## Purpose

Verify whether connections to Thomson Reuters Datastream data servers are valid

## 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
```

## See Also

`close` | `datastream` | `fetch` | `get`

# esig

## Purpose

eSignal Desktop API connection

## Syntax

```
E = esig(user)
```

## Description

`E = esig(user)` creates an eSignal Desktop API connection given the user name `user`. Only one eSignal connection can be open at a time.

## Examples

In order to use the signal interface, you need to make the eSignal Desktop API visible to MATLAB by using the command:

```
% Add NET assembly.
NET.addAssembly('D:\Work\esignal\DesktopAPI_TimeAndSales\...
   DesktopAPI_TimeAndSales\obj\Release\Interop.IESignal.dll');
```

---

**Note** `Interop.IESignal.dll` does not ship with Datafeed Toolbox. This file is created by Microsoft Visual Studio® using an unmanaged DLL, in a managed environment. `Interop.IESignal.dll` is a wrapper that Microsoft Visual Studio creates.

If you do not have `Interop.IESignal.dll`, contact our technical support staff.

Use the `NET.addAssembly` command to access `Interop.IESignal.dll` in MATLAB. For example:

```
NET.addAssembly('D:\Work\esignal\DesktopAPI_TimeAndSales\DesktopAPI_TimeAndSales\obj\Release\Interop.IESi
```

---

Create an eSignal connection handle:

```
% Enter 'mylogin' as your user name.
E = esig('mylogin')
```

## See Also

`close` | `getdata` | `history` | `timeseries`

## Purpose

Close eSignal connection

## Syntax

```
close(e)
```

## Description

`close(e)` closes the eSignal connection object, `e`.

## See Also

`esig`

# getdata

## Purpose

Current eSignal data

## Syntax

```
D = getdata(E,S)
```

## Description

`D = getdata(E,S)` returns the eSignal basic quote data for the security `S`. `E` is a connection object created by `esig`.

## Examples

Return the eSignal basic quote data for the security `ABC`:

```
D = getdata(E,'ABC')
```

## See Also

`esig` | `close` | `history` | `timeseries`

## Purpose

Current eSignal fundamental data

## Syntax

```
D = getfundamentaldata(E,S)
```

## Description

`D = getfundamentaldata(E,S)` returns the eSignal fundamental data for the security `S`.

## Examples

Return the eSignal fundamental data for the security `ABC`:

```
D = getfundamentaldata(E,'ABC')
```

## See Also

`esig` | `close` | `getdata` | `history` | `timeseries`

# history

## Purpose

eSignal historical data

## Syntax

```
D = history(E,S,F,{startdate,enddate},per)
```

## Description

`D = history(E,S,F,{startdate,enddate},per)` returns the historical data for the given inputs. Input arguments include the security list `S`, the fields `F`, the dates `startdate` and `enddate`, and the periodicity `per`. Valid fields are `Time`, `Open`, `High`, `Low`, `Close`, `Volume`, `OI`, `Flags`, `TickBid`, `TickAsk`, and `TickTrade`. The input argument `per` is optional and specifies the period of the data. Possible values for `per` are `'D'` (daily, the default), `'W'` (weekly), and `'M'` (monthly).

## Examples

Return the closing price for the given dates for the given security using the default period of the data:

```
D = history(E,'ABC','CLOSE',{'8/01/2009','8/10/2009'})
```

---

Return the monthly closing and high prices for the given dates for the given security:

```
D = history(E,'ABC',{'close','high'},{'6/01/2009','11/10/2009'},'M')
```

---

Return all fields for the given dates for the given security using the default period of the data. The fields are returned in the following order: `Time`, `Open`, `High`, `Low`, `Close`, `Volume`, `OI`, `Flags`, `TickBid`, `TickAsk`, `TickTrade`.

```
D = history(E,'ABC',[],{'8/01/2009','8/10/2009'})
```

## See Also

`esig` | `close` | `getdata` | `timeseries`

## Purpose

eSignal intraday tick data

## Syntax

```
D = timeseries(E,S,F,{startdate,enddate},per)
D = timeseries(E,S,F,startdate)
```

## Description

`D = timeseries(E,S,F,{startdate,enddate},per)` returns the intraday data for the given inputs. Inputs include the security list `S`, the fields `F`, the dates `startdate` and `enddate`, and the periodicity `per`. Valid fields for `F` are `Time`, `Open`, `High`, `Low`, `Close`, `Volume`, `OI`, `Flags`, `TickBid`, `TickAsk`, and `TickTrade`. The periodicity `per` is optional and specifies the period of the data. For example, if you enter the value `'1'` for `per`, the returned data will be aggregated into 1-minute bars. Enter `'30'` for 30-minute bars and `'60'` for 60-minute bars.

`D = timeseries(E,S,F,startdate)` returns raw intraday tick data for the date range starting at `startdate` and ending with current day. Note that the date range can only extend back for a period of 10 days from the current day.

## Tips

For intraday tick requests made with a period argument, `per`, the following fields are valid: `Time`, `Open`, `High`, `Low`, `Close`, `Volume`, `OI`, `Flags`, `TickBid`, `TickAsk`, and `TickTrade`.

For raw intraday tick requests, the following fields are valid: `TickType`, `Time`, `Price`, `Size`, `Exchange`, and `Flags`.

## Examples

Return the monthly closing and high prices for the given dates for the given security in 10-minute bars.

```
D = timeseries(E,'ABC US Equity',{'close','high'},...
   {'1/01/2010','4/10/2010'},'10')
```

---

Return all fields for the given dates for the given security in 10 minute bars. Fields are returned in the following order: `Time`, `Open`, `High`, `Low`, `Close`, `Volume`, `OI`, `Flags`, `TickBid`, `TickAsk`, and `TickTrade`.

```
D = timeseries(E,'ABC US Equity',[],{'8/01/2009','8/10/2009'},'10')
```

## See Also

esig | close | getdata | history

## Purpose

IQFEED Desktop API connection

## Syntax

```
Q= iqf(username, password)
Q= iqf(username, password, portname)
```

## Description

`Q= iqf(username, password)` starts IQFEED or makes a connection to an existing IQFEED session.

`Q= iqf(username, password, portname)` starts IQFEED or makes a connection to an existing IQFEED session.

**Note** Only one IQFEED connection can be open at a time.

## Arguments

| | |
|---|---|
| `username` | The user name for the IQFEED account. |
| `password` | The password for the IQFEED account. |
| `portname` | The IQFEED port identifier (default = `'Admin'`). |

## Examples

Create an IQFEED connection handle.

```
Q = iqf('username','password')
```

Alternatively, you can create a connection and specify the `portname` argument.

```
Q = iqf('username','password', 'Admin')
```

## See Also

`close` | `history` | `marketdepth` | `news` | `realtime` | `timeseries`

# close

## Purpose

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`

## Purpose

IQFEED asynchronous historical end of period data

## Syntax

```
history(c,s,interval)
history(...,period)
history(...,listener,eventhandler)

history(c,s,{startdate,enddate})
history(...,[],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(...,period)` returns asynchronous historical end of period data for a single security with a specified interval and period `period`.

`history(...,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(...,[],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`.

## Input Arguments

### c - IQFEED connection

connection object

IQFEED connection, specified as a connection object created using `iqf`.

### s - Security

string

Security, specified as a string for a single security.

**Example:** `'IBM'`

**Data Types**
`char`

### interval - Time interval
scalar

Time interval, specified as a 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 above enumerated strings 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.

**Data Types**
`char`

### 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.

**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 MATLAB Command Window. You can modify the existing event handler function or define your own.

**Data Types**
`function_handle`

#### startdate - Start date
scalar | string

Start date, specified as a scalar or string to denote the start date of the date range for the returned data.

**Example:** `floor(now-1)`

**Data Types**
`double | char`

#### enddate - End date
scalar | string

End date, specified as a scalar or string to denote the end date of the date range for the returned data.

**Example:** `floor(now)`

**Data Types**
`double | char`

## 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
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033
```

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'    '1
    '2013-11-15 11:07:02'    '1039.75'    '1005.00'    '1009.51'    '1
    '2013-11-08 11:07:02'    '1032.37'    '1007.64'    '1031.50'    '1
    '2013-11-01 11:07:02'    '1041.52'    '1012.98'    '1015.20'    '1
    '2013-10-25 11:07:02'    '1040.57'    '995.79'     '1011.46'    '1
```

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
    '2013-10-31 11:13:07'    '1041.52'    '842.98'     '880.25'     '1030
    '2013-09-30 11:13:07'    '905.99'     '853.95'     '854.36'     '875.
    '2013-08-30 11:13:07'    '909.71'     '845.56'     '895.00'     '846.
    '2013-07-31 11:13:07'    '928.00'     '875.61'     '886.45'     '887.
```

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'
    '2013-11-20 10:59:51'    '186.2400'    '184.6450'    '185.2200'
    '2013-11-19 10:59:51'    '186.2000'    '184.1500'    '184.6300'
    '2013-11-18 10:59:51'    '184.9900'    '183.2700'    '183.5200'
```

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'    '1
    '2013-11-20 11:12:15'    '1033.36'    '1020.36'    '1029.95'    '1
    '2013-11-19 11:12:15'    '1034.75'    '1023.05'    '1031.72'    '1
    '2013-11-18 11:12:15'    '1048.74'    '1029.24'    '1035.75'    '1
```

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);
```

## See Also

`iqf` | `close` | `marketdepth` | `realtime` | `timeseries`

# marketdepth

## Purpose

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 string for a single security or a cell array of strings 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,@iqfeedmarketdeptheventhandler)
openvar('IQFeedLevelTwoData')
```

## See Also

iqf | close | history | realtime | timeseries

## Purpose

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 string for a single security or a cell array of strings 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

iqf | close | history | marketdepth | realtime | timeseries

# realtime

## Purpose

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.

## Arguments

| | |
|---|---|
| `Q` | IQFEED connection handle created using `iqf.` |
| `S` | `S` is specified as a string for a single security or a cell array of strings 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 in the 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 in the variable `IQFeedLevelOneData`.

```
realtime(q,'ABC',...
{'Symbol','Exchange ID','Last','Change','Incremental Volume'})
openvar('IQFeedLevelOneData')
```

---

Return level 1 data for security ABC using the function handles `iqfeedlistener` and `iqfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedLevelOneData`.

```
realtime(q,'ABC',...
      {'Symbol','Exchange ID','Last','Change','Incremental Volume'},..
      @iqfeedlistener,@iqfeedeventhandler)
    openvar('IQFeedLevelOneData')
```

## See Also

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

# timeseries

## Purpose

IQFEED asynchronous historical end of period 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 string. |
| `daterange` | Ether 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 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. |

## Examples

Return intraday ticks for a given date range and use the default socket listener and event handler and then 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 Time Stamp, Last, Last Size, Total Volume, Bid, Ask, and TickID.

---

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 Request ID, Time, Stamp, High, Low, Open, Close, Total Volume, and 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 `iqfeedlistener` and `iqfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{floor(now),now},[],@iqtimeseriesfeedlistener,@iqtimeseriesfeedeventhandler)
openvar('IQFeedTimeseriesData')
```

## See Also

`iqf` | `close` | `history` | `marketdepth` | `realtime`

# factset

## Purpose

Establish connection to FactSet data

## Syntax

```
Connect = factset('UserName', 'SerialNumber', 'Password', 'ID')
```

## Arguments

| | |
|---|---|
| `UserName` | User login name. |
| `SerialNumber` | User serial number. |
| `Password` | User password. |
| `ID` | FactSet customer identification number. |

**Note** FactSet assigns values to all input arguments.

## Description

`Connect = factset('UserName', 'SerialNumber', 'Password', 'ID')` connects to the FactSet interface.

## Examples

Establish a connection to FactSet data:

```
Connect = factset('username', '1234', 'password', 'fsid')
Connect =
       user: 'username'
     serial: '1234'
   password: 'password'
        cid: 'fsid'
```

## See Also

`close` | `fetch` | `get` | `isconnection`

## Purpose

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`

# fetch

## Purpose

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 string or cell array of strings containing the names of securities in a format recognizable by the FactSet server. |
| `Fields` | A MATLAB string or cell array of strings indicating the data fields for which to retrieve data. |
| `Date` | Date 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. |

**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`

## Purpose

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 string or cell array of strings 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` | `fetch` | `factset` | `isconnection`

## Purpose

Verify whether 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` | `fetch` | `factset` | `get`

# fds

## Purpose

Create FactSet Data Server connection

## Syntax

```
Connect = fds(UserName,Password)
Connect = fds(UserName,Password,Finfo)
```

## Description

`Connect = fds(UserName,Password)` connects to the FactSet Data Server or local workstation using the field information file, `rt_fields.xml`, found on the MATLAB path. The file `rt_fields.xml` can be obtained from FactSet.

`Connect = fds(UserName,Password,Finfo)` connects to the FactSet Data Server or local workstation using the specified field information file (`Finfo`).

## Input Arguments

### UserName - User login name

string

User login name to FactSet Data Server, specified as a string.

**Data Types**

`char`

### Password - User password

string

User password to FactSet Data Server, specified as a string.

**Data Types**

`char`

### Finfo - Field information

string

Field information, specified as a string.

**Example:** `'C:\Program Files (x86)\FactSet\FactSetDataFeed\fdsrt-2\etc\rt_fields.xml'`

**Data Types**
`char`

## Output Arguments

### Connect - Connection object

object structure

Connection object for FactSet Data Server, returned as an object for class `FDS`.

## Examples

### Create FDS Connection

Connect to the FactSet Data Server.

```
f = fds('USER','123456');
```

This creates the connection object `C` using the field information file, `rt_fields.xml`, found on the MATLAB path. You can obtain the file `rt_fields.xml` from FactSet.

### Create FDS Connection Using `Finfo`

Connect to the FactSet Data Server using the optional `Finfo` input argument.

```
f = fds('USER','123456',...
'C:\Program Files (x86)\FactSet\FactSetDataFeed\fdsrt-2\etc\rt_fields.xml');
```

This creates the connection object `C`.

## See Also

`close` | `realtime` | `stop`

## Purpose

Obtain real-time data from FactSet Data Server

## Syntax

```
T = realtime(F,Srv,Sec,Cb)
T = realtime(F,Srv,Sec)
```

## Description

`T = realtime(F,Srv,Sec,Cb)` asynchronously requests real-time or streaming data from the FactSet Data Server or local workstation.

`T = realtime(F,Srv,Sec)` asynchronously requests real-time or streaming data from the FactSet Data Server or local workstation. When `Cb` is not specified, the default message event handler `factsetMessageEventHandler` is used.

## Input Arguments

### F - FactSet Data Server connection object

object structure

FactSet Data Server connection object, specified using `fds`.

### Srv - Data source or supplier

string

Data source or supplier, specified as a string.

**Example:** `'FDS1'`

**Data Types**
`char`

### Sec - Security symbol

string

Security symbol, specified as a string.

**Example:** `'ABCD-USA'`

**Data Types**
`char`

### Cb - Event handler

function handle

Event handler, specified as a function handle requests real-time or streaming data from the service FactSet Data Server.

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 Data Server.

## Examples

### Request FactSet Data Server 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 = f.realtime('FDS1','ABCD-USA',@(varargin)myMessageEventHandler(varargin))
```

### Request FactSet Data Server 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 = f.realtime('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.

## See Also

`fds` | `close` | `stop`

# stop

## Purpose

Cancel real-time request

## Syntax

```
stop(F,T)
```

## Description

`stop(F,T)` cancels a real-time request. This function cleans up resources associated with real-time requests that are no longer needed.

## Input Arguments

### `F` - Connection object

object structure

Connection object, specified using `fds`.

### `T` - Real-time request tag

nonnegative integer

Real-time request tag, specified using `realtime`.

**Data Types**

`double`

## Examples

### Cancel FactSet Data Server Real-Time Request

Terminate a FactSet Data Server real-time request.

```
T = f.realtime('FDS1','GOOG-USA')
f.stop(T)
```

## See Also

`fds` | `close` | `realtime`

## Purpose

Disconnect from FactSet Data Server

## Syntax

```
close(F)
```

## Description

`close(F)` disconnects from the FactSet Data Server or local workstation given the connection object, `F`.

## Input Arguments

### `F` - Connection object

object structure

Connection object, specified using `fds`.

## Examples

### Close FactSet Data Server Connection

Close the FactSet Data Server connection.

```
F = f.realtime('FDS1','GOOG-USA')
close(F)
```

## See Also

`fds` | `realtime` | `stop`

# fred

## Purpose

Connect to FRED data servers

## Syntax

```
Connect = fred(URL)
Connect = fred
```

## Arguments

| | |
|---|---|
| `URL` | Create a connection using a specified URL. |

## Description

`Connect = fred(URL)` establishes a connection to a FRED data server.

`Connect = fred` verifies that the URL `http://research.stlouisfed.org/fred2/` is accessible and creates a connection.

## Examples

Connect to the FRED data server at the URL `http://research.stlouisfed.org/fred2/`:

```
c = fred('http://research.stlouisfed.org/fred2/')
```

## See Also

`close` | `fetch` | `get` | `isconnection`

## Purpose

Close connections to FRED data servers

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FRED connection object created with `fred`. |

## Description

`close(Connect)` closes the connection to the FRED data server.

## Examples

Make a connection `c` to a FRED data server:

```
c = fred('http://research.stlouisfed.org/fred2/')
```

Close this connection:

```
close(c)
```

## See Also

`fred`

# fetch

## Purpose

Request data from FRED data servers

## Syntax

```
data = fetch(Connect, 'Series')
data = fetch(Connect, 'Series', 'D1')
data = fetch(Connect, 'Series', 'D1', 'D2')
```

## Arguments

| | |
|---|---|
| `Connect` | FRED connection object created with the `fred` function. |
| `'Series'` | MATLAB string containing the name of a series in a format recognizable by the FRED server. |
| `'D1'` | MATLAB string or date number indicating the date from which to retrieve data. |
| `'D2'` | MATLAB string or date number indicating the date range from which to retrieve data. |

## Description

For a given series, `fetch` returns historical data using the connection to the FRED data server.

`data = fetch(Connect, 'Series')` returns data for `Series`, using the connection object `Connect`.

`data = fetch(Connect, 'Series', 'D1')` returns data for `Series`, using the connection object `Connect`, for the date `D1`.

`data = fetch(Connect, 'Series', 'D1', 'D2')` returns all data for `Series`, using the connection object `Connect`, for the date range `'D1'` to `'D2'`.

**Note** You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

## Examples

Fetch all available daily U.S. dollar to European foreign exchange rates.

```
d = fetch(f,'DEXUSEU')
d =
                Title: 'U.S. / Euro Foreign Exchange Rate'
             SeriesID: 'DEXUSEU'
               Source: 'Board of Governors of the Federal Reserve System'
              Release: 'H.10 Foreign Exchange Rates'
    SeasonalAdjustment: 'Not Applicable'
            Frequency: 'Daily'
                Units: 'U.S. Dollars to One Euro'
            DateRange: '1999-01-04 to 2006-06-19'
          LastUpdated: '2006-06-20 9:39 AM CT'
                Notes: 'Noon buying rates in New York City for
                         cable transfers payable in foreign currencies.'
                 Data: [1877x2 double]
```

`Data` is an N-by-2 element double array that contains dates in the first column and the series values in second column.

Fetch data for 01/01/2007 through 06/01/2007.

```
d = fetch(f, 'DEXUSEU', '01/01/2007', '06/01/2007')
d =
                Title: ' U.S. / Euro Foreign Exchange Rate'
             SeriesID: ' DEXUSEU'
               Source: ' Board of Governors of the Federal Reserve System'
              Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Applicable'
            Frequency: ' Daily'
                Units: ' U.S. Dollars to One Euro'
            DateRange: ' 1999-01-04 to 2006-06-19'
          LastUpdated: ' 2006-06-20 9:39 AM CT'
                Notes: ' Noon buying rates in New York City for
                         cable transfers payable in foreign currencies.'
                 Data: [105x2 double]
```

`Data` is an N-by-2 element double array that contains dates in the first column and the series values in second column.

## See Also

`close` | `get` | `isconnection`

## Purpose

Retrieve properties of FRED connection objects

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FRED connection object created with `fred`. |
| `'PropertyName'` | A MATLAB string or cell array of strings containing property names. Property names are:<br>• `'url'`<br>• `'ip'`<br>• `'port'` |

## Description

`value = get(Connect, 'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the FRED connection object.

`value = get(Connect)` returns the value for all properties.

## Examples

Establish a connection, `c`, to a FRED data server:

```
c = fred('http://research.stlouisfed.org/fred2/')
```

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`

# isconnection

## Purpose

Verify whether connections to FRED data servers are valid

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | FRED connection object created with `fred`. |

## Description

`x = isconnection(Connect)` returns `x = 1` if a connection to the FRED data server is valid, and `x = 0` otherwise.

## Examples

Establish a connection, `c`, to a FRED data server:

```
c = fred('http://research.stlouisfed.org/fred2/')
```

Verify that `c` is a valid connection:

```
x = isconnection(c)
x = 1
```

## See Also

`close` | `fetch` | `get`

## Purpose

Connect to local Haver Analytics database

## Syntax

```
H = haver(Databasename)
```

## Arguments

| | |
|---|---|
| `Databasename` | Local path to the Haver Analytics database. |

## Description

`H = haver(Databasename)` establishes a connection to a Haver Analytics database.

---

**Requirement:** Both read and write permissions are required on the database file to establish a database connection. Otherwise, this error message appears: `Unable to open specified database file`.

---

## Examples

Create a connection to the Haver Analytics database at the path `d:\work\haver\data\haverd.dat`:

```
H = haver('d:\work\haver\data\haverd.dat')
```

## See Also

`close` | `fetch` | `get` | `isconnection`

# aggregation

## Purpose

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

`haver` | `close` | `fetch` | `get` | `info` | `isconnection` | `nextinfo`

## Purpose

Close Haver Analytics database

## Syntax

```
close(H)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with `haver`. |

## Description

`close(H)` closes the connection to the Haver Analytics database.

## Examples

Establish a connection `H` to a Haver Analytics database:

```
H = haver('d:\work\haver\data\haverd.dat')
```

Close the connection:

```
close(H)
```

## See Also

`haver`

# fetch

## Purpose

Request data from Haver Analytics database

## Syntax

```
D = fetch(H,S)
D = fetch(H,S,Startdate,Enddate)
D = fetch(H,S,Startdate,Enddate,P)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with`haver`. |
| `S` | Haver Analytics variable. |
| `Startdate` | MATLAB string or date number indicating the `startdate` from which to retrieve data. |
| `Enddate` | MATLAB string or date number indicating the `enddate` of the date range. |
| `P` | A specified period. You can enter the period as:<br>• `D` for daily values<br>• `W` for weekly values<br>• `M` for monthly values<br>• `Q` for quarterly values<br>• `A` for annual values |

## Description

`fetch` returns historical data via a Haver Analytics connection object.

`D = fetch(H,S)` returns data for the Haver Analytics variable `S`, using the connection object `H`.

`D = fetch(H,S,Startdate,Enddate)` returns data for the Haver Analytics variable `S`, using the connection object `H`, between the dates `Startdate` and `Enddate`.

`D = fetch(H,S,Startdate,Enddate,P)` returns data for the Haver Analytics variable `S`, using the connection object `H`, between the dates `Startdate` and `Enddate`, in time periods specified by `P`.

## Examples

### Establish a Connection to a Haver Analytics Database

Connect to the Haver Analytics daily demonstration database `haverd.dat`:

```
H = haver('d:\work\haver\data\haverd.dat')
```

### Retrieving Variable Data

Return data for the variable `FFED`:

```
D = fetch(H,'FFED')
```

### Retrieving Variable Data for a Specified Date Range

Return data for `FFED` from 01/01/1997 to 09/01/2007:

```
D = fetch(H,'FFED','01/01/1997','09/01/2007')
```

### Retrieving Monthly Variable Data for a Specified Date Range

Return data for `FFED`, converted to monthly values, from 01/01/1997 to 09/01/2007:

```
D = fetch(H,'FFED','01/01/1997','09/01/2007','M')
```

## See Also

`close` | `get` | `isconnection` | `haver` | `info` | `nextinfo`

## Purpose

Retrieve properties from Haver Analytics connection objects

## Syntax

```
V = get(H,'PropertyName')
V = get(H)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with `haver`. |
| `'PropertyName'` | A MATLAB string or cell array of strings containing property names. The property name is `Databasename`. |

## Description

`V = get(H,'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the Haver Analytics connection object.

`V = get(H)` returns a MATLAB structure, where each field name is the name of a property of `H`. Each field contains the value of the property.

## Examples

Establish a Haver Analytics connection, `HDAILY`:

```
HDAILY = haver('d:\work\haver\data\haverd.dat')
```

Retrieve the name of the Haver Analytics database:

```
V = get(HDAILY,{'databasename'})
V=
databasename: d:\work\haver\data\haverd.dat
```

## See Also

`close` | `fetch` | `isconnection` | `haver`

## Purpose

Retrieve information about Haver Analytics variables

## Syntax

```
D = info(H,S)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with `haver`. |
| `S` | Haver Analytics variable. |

## Description

`D = info(H,S)` returns information about the Haver Analytics variable, `S`.

## Examples

Establish a Haver Analytics connection `H`:

```
H = haver('d:\work\haver\data\haverd.dat')
```

Request information for the variable after `FFED`:

```
D = info(H,'FFED2')
```

The following output is returned:

```
     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'
```

## See Also

close | get | isconnection | haver | nextinfo

## Purpose

Verify whether connections to Haver Analytics data servers are valid

## Syntax

```
X = isconnection(H)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with `haver`. |

## Description

`X = isconnection(H)` returns `X = 1` if the connection is a valid Haver Analytics connection, and `X = 0` otherwise.

## Examples

Establish a Haver Analytics connection `H`:

```
H = HAVER('d:\work\haver\data\haverd.dat')
```

Verify that `H` is a valid Haver Analytics connection:

```
X = isconnection(H)
X = 1
```

## See Also

`close` | `fetch` | `get` | `haver`

# nextinfo

## Purpose

Retrieve information about next Haver Analytics variable

## Syntax

```
D = nextinfo(H,S)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with the `haver` function. |
| `S` | Haver Analytics variable. |

## Description

`D = nextinfo(H,S)` returns information for the next Haver Analytics variable after the variable, `S`.

## Examples

Establish a Haver Analytics connection `H`:

```
H = haver('d:\work\haver\data\haverd.dat')
```

Request information for the variable following `FFED`:

```
D = nextinfo(H,'FFED')
```

The following structure is returned:

```
     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'
```

## See Also

close | get | haver | info | isconnection

# havertool

## Purpose

Run Haver Analytics graphical user interface (GUI)

## Syntax

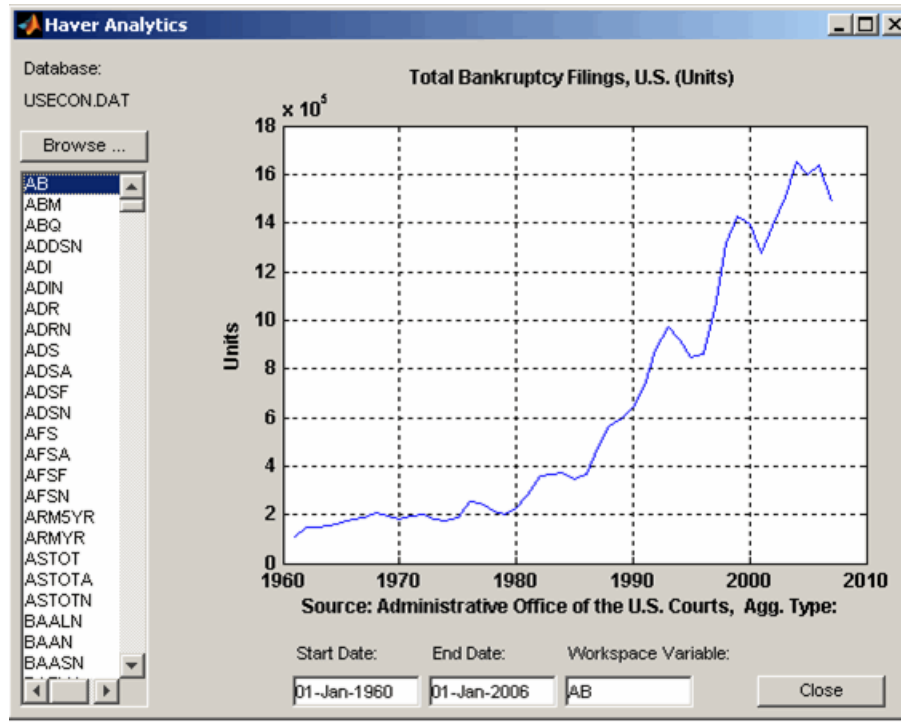
```
havertool(H)
```

## Arguments

| | |
|---|---|
| `H` | Haver Analytics connection object created with `haver`. |

## Description

`havertool(H)` 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`:

```
H = haver('d:\work\haver\data\haverd.dat')
```

Open the graphical user interface (GUI) demonstration:

```
havertool(H)
```

## See Also

`haver`

# idc

## Purpose

Connect to Interactive Data data servers

## Syntax

```
Connect = idc
```

## Description

`Connect = idc` connects to the Interactive Data server. `Connect` is a connection handle used by other functions to obtain data.

## Examples

Connect to an Interactive Data server:

```
c = idc
```

## See Also

`close` | `fetch` | `get` | `isconnection`

## Purpose

Close connections to Interactive Data data servers

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Interactive Data connection object created with `idc`. |

## Description

`close(Connect)` closes the connection to the Interactive Data server.

## Examples

Establish an Interactive Data connection, `c`:

```
c = idc
```

Close this connection:

```
close(c)
```

## See Also

`idc`

# fetch

## Purpose

Request data from Interactive Data data servers

## Syntax

```
data = fetch(Connect, 'Security', 'Fields')
data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')
data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate',
   'Period')
data = fetch(Connect,'','GUILookup','GUICategory')
```

## Arguments

| | |
|---|---|
| `Connect` | Interactive Data connection object created with `idc`. |
| `'Security'` | A MATLAB string containing the name of a security in a format recognizable by the Interactive Data server. |
| `'Fields'` | A MATLAB string or cell array of strings indicating specific fields for which to provide data. Valid field names are in the file `@idc/idcfields.mat`. The variable `bbfieldnames` contains the list of field names. |
| `'FromDate'` | Beginning date for historical data. |

**Note** You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.

| | |
|---|---|
| `'ToDate'` | End date for historical data. |
| `'Period'` | Period within date range. |
| `'GUICategory'` | GUI category. Possible values are:<br>• `'F'` (All valid field categories)<br>• `'S'` (All valid security categories) |

## Description

`data = fetch(Connect, 'Security', 'Fields')` returns data for the indicated fields of the designated securities. Load the file `idc/idcfields` to see the list of supported fields.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')` returns historical data for the indicated fields of the designated securities.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period')` returns historical data for the indicated fields of the designated securities with the designated dates and period. Consult the Remote Plus documentation for a list of valid `'Period'` values.

`data = fetch(Connect,'','GUILookup','GUICategory')` opens the Interactive Data dialog box for selecting fields or securities.

## Examples

Open the dialog box to look up securities:

```
D = fetch(Connect,'','GUILookup','S')
```

Open the dialog box to select fields:

```
D = fetch(Connect,'','GUILookup','F')
```

## See Also

`close` | `get` | `idc` | `isconnection`

## Purpose

Retrieve properties of Interactive Data connection objects

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Interactive Data connection object created with `idc`. |
| `PropertyName` | (Optional) A MATLAB string or cell array of strings containing property names. Property names are:<br>• `'Connected'`<br>• `'Connection'`<br>• `'Queued'` |

## Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the Interactive Data connection object.`PropertyName` is a string or cell array of strings containing property names.

`value = get(Connect)` returns a MATLAB structure. Each field name is the name of a property of `Connect`, and each field contains the value of that property.

## See Also

`close` | `idc` | `isconnection`

## Purpose

Verify whether connections to Interactive Data data servers are valid

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Interactive Data connection object created with `idc`. |

## Description

`x = isconnection(Connect)` returns `x = 1` if the connection is a valid Interactive Data connection, and `x = 0` otherwise.

## Examples

Establish an Interactive Data connection `c`:

```
c = idc
```

Verify that `c` is a valid connection:

```
x = isconnection(c)
x = 1
```

## See Also

`close` | `fetch` | `get` | `idc`

## Purpose

Connect to Kx Systems, Inc. kdb+ databases

## Syntax

```
k = kx(ip,p)
k = kx(ip,p,id)
```

## Arguments

| | |
|---|---|
| `ip` | IP address for the connection to the Kx Systems, Inc. kdb+ database. |
| `p` | Port for the Kx Systems, Inc. kdb+ database connection. |
| `id` | The *username:password* string for the Kx Systems, Inc. kdb+ database connection. |

## Description

`k = kx(ip,p)` connects to the Kx Systems, Inc. kdb+ database given the IP address `ip` and port number `p`.

`k = kx(ip,p,id)` connects to the Kx Systems, Inc. kdb+ database given the IP address `ip`, port number `p`, and *username:password* string `id`.

Before you connect to the database, add The Kx Systems, Inc. file `jdbc.jar` to the MATLAB `javaclasspath` using the `javaaddpath` command. The following example adds `jdbc.jar` to the MATLAB `javaclasspath` `c:\q\java`:

```
javaaddpath c:\q\java\jdbc.jar
```

---

**Note** In earlier versions of the Kx Systems, Inc. kdb+ database, this jar file was named `kx.jar`. If you are running an earlier version of the database, substitute `kx.jar` for `jdbc.jar` in these instructions to add this file to the MATLAB `javaclasspath`.

---

## 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)
handle: [1x1 c]
        ipaddress: 'localhost'
        port: 5001
```

## See Also

`close` | `exec` | `get` | `fetch` | `tables`

# close

## Purpose

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`

## Purpose

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`

## Purpose

Request data from Kx Systems, Inc. kdb+ databases

## Syntax

```
d = fetch(k,ksql)
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 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,p2,p3)` executes the command specified in `ksql` with one or more input parameters, and returns the data from this command.

## 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 `select from trade`:

```
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`

## Purpose

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 string or cell array of strings 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`

# insert

## Purpose

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`

## Purpose

Verify whether 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`

# tables

## Purpose

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`

## Purpose

Connect to Thomson Reuters Tick History

## 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 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.TRTHAp
           user: 'user@company.com'
       password: '**********'
           cred: [1x1 com.thomsonreuters.tickhistory.webservice.types.
    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)
```

## See Also

`close` | `fetch` | `get`

## Purpose

Close Thomson Reuters Tick History connection

## Syntax

```
close(r)
```

## Description

`close(r)` closes the Thomson Reuters Tick History connection, `r`.

## See Also

`rdth`

# fetch

## Purpose

Request Thomson Reuters Tick History data

## 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,marketdepth)
```

## 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 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` 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.

---

## Tips

- To obtain more information request and message types and their associated field lists, use the command `get(r)`.

## 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.

For a `'NasdaqLevel2'` request type, enter:

```
AaplTickData = fetch(R,'AAPL.O',{'Nominal Value'},...
     {now-.05,now},'NasdaqLevel2','Nominal Value','NSQ','EQU');
```

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);
```

## See Also

`rdth` | `close` | `get`

## Purpose

Get Thomson Reuters Tick History connection properties

## 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 string or cell array of strings containing 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

rdth | fetch

## Purpose

Verify whether Thomson Reuters Tick History connections are valid

## 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

`rdth` | `close` | `fetch` | `get`

# status

## Purpose

Status of FTP request for Thomson Reuters Tick History data

## 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.

## Purpose

Submit FTP request for Thomson Reuters Tick History data

## 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');
```

For a `'NasdaqLevel2'` request type, enter:

```
AaplTickData = submitftp(R,'AAPL.O',{'Nominal Value'},...
      {now-.05,now},'NasdaqLevel2','Nominal Value','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`

## Purpose

Retrieve data from Thomson Reuters Tick History file

## 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 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 `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

# reuters

## Purpose

Create Reuters sessions

## Syntax

```
r = reuters (sessionName, serviceName)
r = reuters (sessionName, serviceName, user, position)
```

## Arguments

| | |
|---|---|
| `r` | Reuters session object created with `reuters`. |
| `sessionName` | Name of the Reuters session, of the form `myNameSpace::mySession`. |
| `serviceName` | Name of the service you use to connect to the data server. |
| `user` | User ID you use to connect to the data server. |
| `position` | IP address of the data server to which you connect to retrieve data. |

## Description

You must configure your environment before you use this function to connect to a Reuters data server. For details, see "Reuters Data Service Requirements" on page 1-5.

`r = reuters (sessionName, serviceName)` starts a Reuters session where `sessionName` is of the form `myNameSpace::mySession` and `serviceName` specifies the name of the service you use to connect to the data server.

`r = reuters (sessionName, serviceName, user, position)` starts a Reuters session where `sessionName` is of the form `myNameSpace::mySession` and `serviceName` is the service to use, `user` is the user ID, and `position` is the IP address of the machine to which you connect to retrieve data. Use this form of the command if you require DACS authentication.

## Examples

### Connecting to Reuters Data Servers

Connect to a Reuters data server with session name `'myNS::remoteSession'` and service name `'dIDN_RDF'`:

```
r = reuters ('myNS::remoteSession', 'dIDN_RDF')
r =
session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
user: []
serviceName: 'dIDN_RDF'
standardPI:
[1x1 com.reuters.rfa.common.StandardPrincipalIdentity]
eventQueue: [Error]
marketDataSubscriber:
[1x1 com.reuters.rfa.internal.session.
MarketDataSubscriberImpl]
marketDataSubscriberInterestSpec:
[1x1 com.reuters.rfa.session.MarketDataSubscriber
InterestSpec]
client:
[1x1 com.mathworks.toolbox.datafeed.MatlabReutersClient]
mdsClientHandle:
[1x1 com.reuters.rfa.internal.common.HandleImpl]
```

---

**Note** If you do not use the Reuters DACS authentication functionality, the following error message appears:

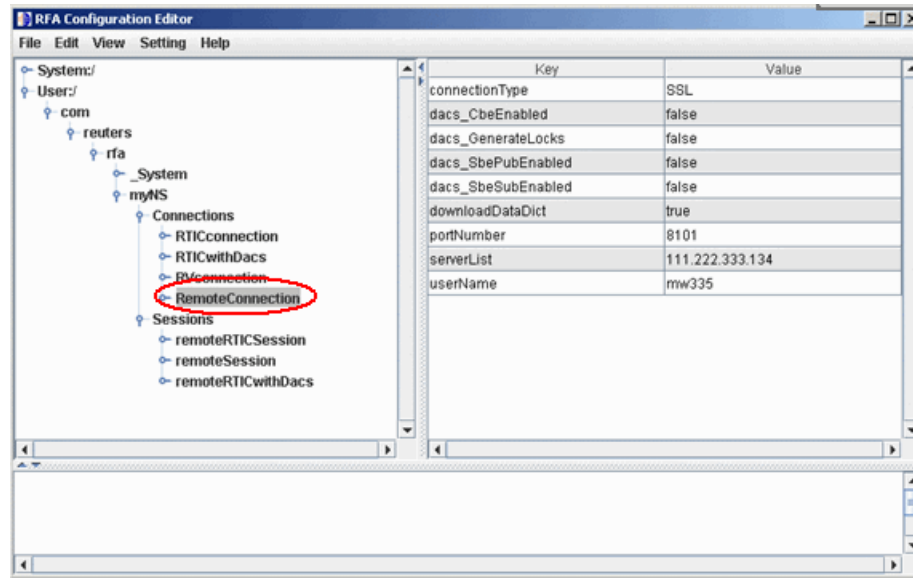
```
com.reuters.rfa.internal.connection.ConnectionImpl
initializeEntitlementsINFO:
com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

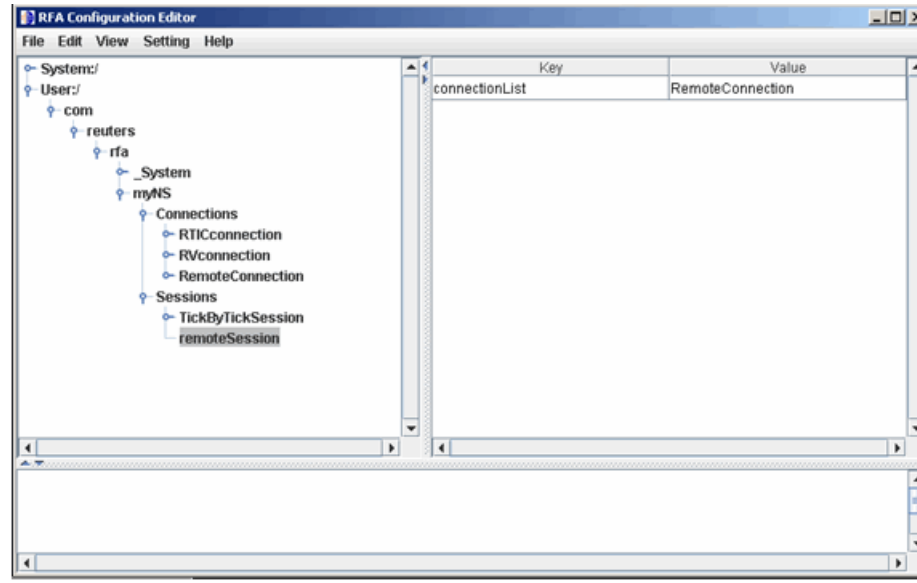
---

### Connecting to Reuters Data Servers Using DACS Authentication

1. Connect to a Reuters data server using DACS authentication, with session name `'myNS::remoteSession'`, service name `'dIDN_RDF'`, user id `'ab123'`, and data server IP address `'111.222.333.444/net'`:

```
r = reuters ('myNS::remoteSession', 'dIDN_RDF', ...
'ab123', '111.222.333.444/net')
```






2 Add the following to your connection configuration:

```
dacs_CbeEnabled=false
dacs_SbePubEnabled=false
dacs_SbeSubEnabled=false
```

3 If you are running an SSL connection, add the following to your connection configuration:

```
dacs_GenerateLocks=false
```

## Connecting to Reuters Data Servers Without DACS Authentication

Connect to a Reuters data server with session name `'myNS::remoteSession'` and service name `'dIDN_RDF'`, without using DACS:

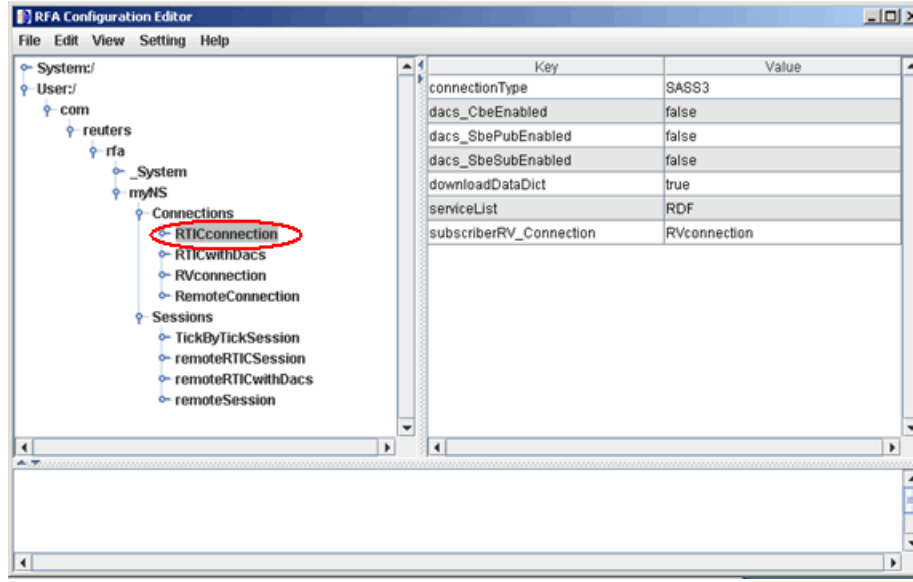
```
r = reuters ('myNS::remoteSession', 'dIDN_RDF')
```

### Establishing an RTIC (TIC-RMDS Edition) Connection to Reuters Data Servers

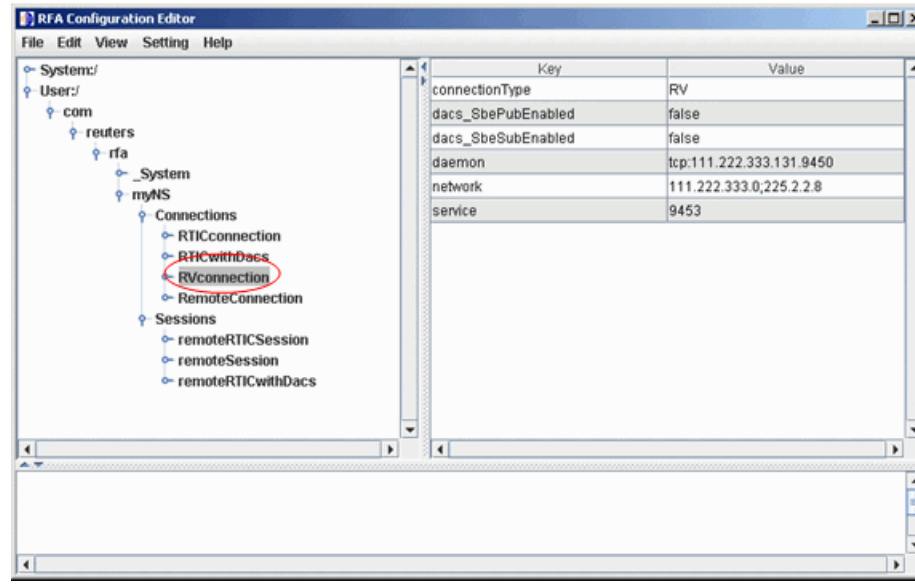
- **Non-DACs-enabled**

  Make an RTIC (TIC-RMDS Edition) connection to a Reuters data server without DACS authentication, with session name `'myNS::remoteRTICSession'`, service name `'IDN_RDF'`:

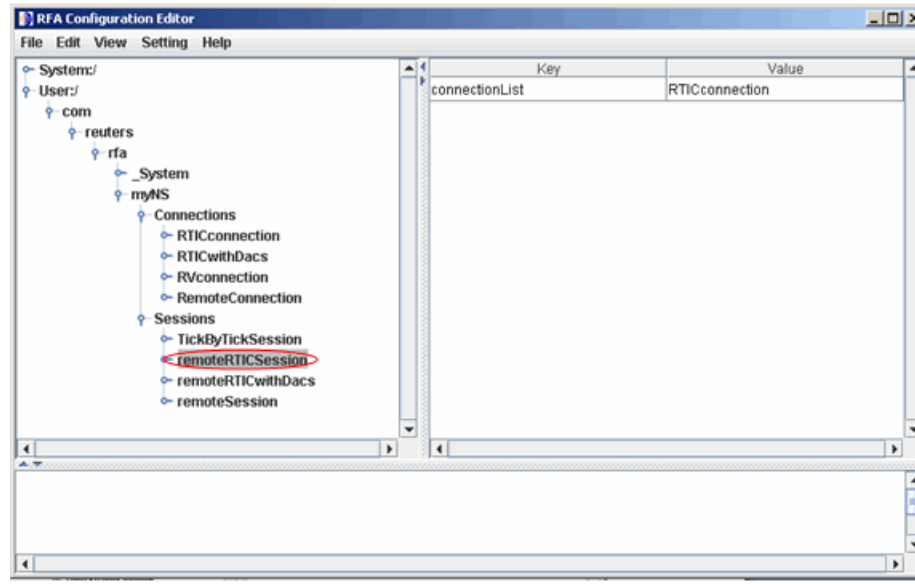
  ```
  r = reuters ('myNS::remoteRTICSession','IDN_RDF')
  ```




This RTIC connection depends on the key subscriber `RVConnection`. Your `RVConnection` configuration should look as follows:




The `RTICConnection` configuration is referenced by the session `remoteRTICSession`, as shown in the following figure.


RFA Configuration Editor
File Edit View Setting Help
System:/
User:/
com
reuters
rfa
_System
myNS
Connections
RTICconnection
RTICwithDacs
RVconnection
RemoteConnection
Sessions
TickByTickSession
remoteRTICSession
remoteRTICwithDacs
remoteSession
Key
Value
connectionList
RTICconnection

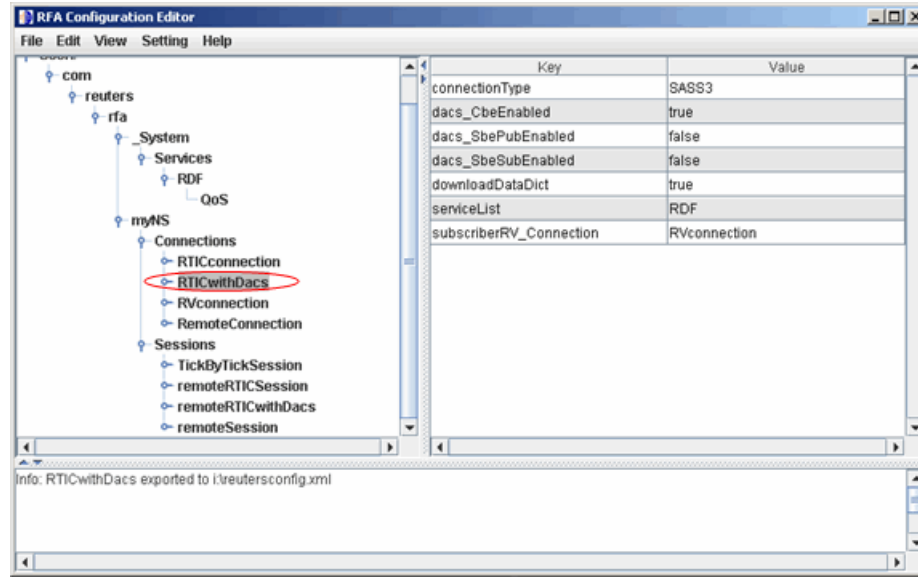
Messages like the following may appear in the MATLAB Command Window when you establish a non-DACs-enabled connection. These messages are informational and can safely be ignored.

```
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
```

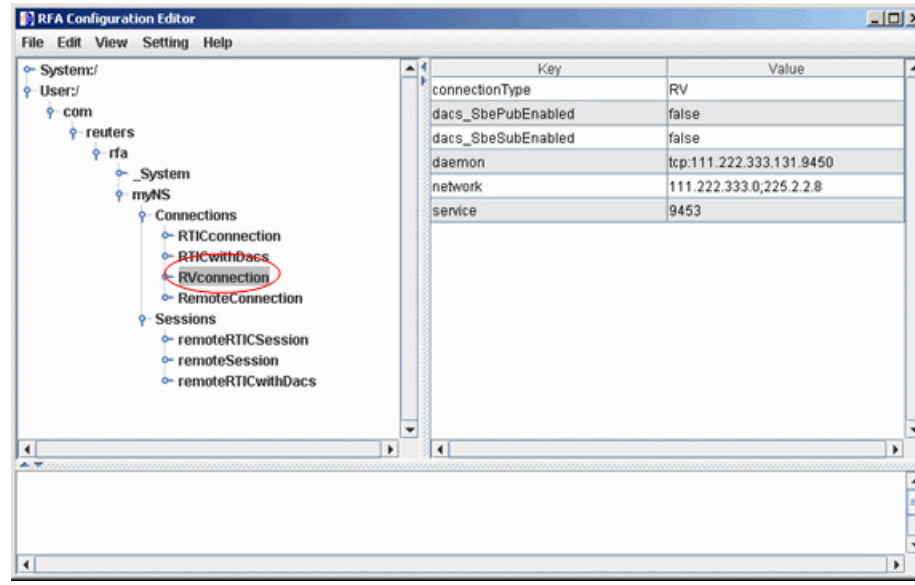
- **DACs-enabled**

  Make an RTIC (TIC-RMDS Edition), DACS-enabled connection to a Reuters data server, with session name `'myNS::remoteRTICWithDACs'`, service name `'IDN_RDF'`, user id `'ab123'`, and data server IP address `'111.222.333.444/net'`:

  ```
  r = reuters ('myNS::remoteRTICWithDACs', 'IDN_RDF', ...
  'ab123', '111.222.333.444/net')
  ```




This RTIC connection depends on the key subscriber `RVConnection`. Your `RVConnection` configuration should look as follows:




Messages like the following may appear in the MATLAB Command Window when you establish a DACs-enabled connection. These messages are informational and can be ignored safely.

```
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
```

If messages like the following appear in the MATLAB Command Window when you establish a DACs-enabled connection:

```
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 the following files:

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

## See Also

`addric` | `close` | `contrib` | `deleteric` | `fetch` | `get` | `history` | `stop` | `rmdsconfig`

## Purpose

Create Reuters Instrument Code

## Syntax

```
addric(r,ric,fid,fval,type)
```

## Description

`addric(r,ric,fid,fval,type)` creates a Reuters Instrument Code, `ric`, on the service defined by the Reuters session, `r`. 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(r,'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(r,'myric',{6,22},{0,0},'live')
```

## See Also

`reuters` | `contrib` | `deleteric` | `fetch`

# close

## Purpose

Release connections to Reuters data servers

## Syntax

```
close(r)
```

## Arguments

| | |
|---|---|
| `r` | Reuters session object created with `reuters`. |

## Description

`close(r)` releases the Reuters connection `r`.

## Examples

Release the connection `r` to the Reuters data server, and unsubscribe all requests associated with it:

```
close(r)
```

## See Also

`reuters`

## Purpose

Contribute data to Reuters data feed

## Syntax

```
contrib(r,s,fid,fval)
```

## Description

`contrib(r,s,fid,fval)` contributes data to a Reuters data feed. `r` 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 `r` and the RIC `'myric'`. Provide a last trade price of 33.5.

```
contrib(r,'myric','trdprc_1',33.5)
```

---

Contribute an additional bid price of 33.8:

```
contrib(r,'myric',{'trdprc_1','bid'},{33.5,33.8})
```

---

Submit value 33.5 for field 6 (`'trdprc_1'`):

```
contrib(r,'myric',6,33.5)
```

---

Add the value 33.8 to field 22 (`'bid'`):

```
contrib(r,'myric',{6,22},{33.5,33.8})
```

## See Also

`reuters` | `addric` | `deleteric` | `fetch`

# deleteric

## Purpose

Delete Reuters Instrument Code

## Syntax

```
deleteric(r,ric)
deleteric(r,ric,fid)
```

## Description

`deleteric(r,ric)` deletes the Reuters Instrument Code, `ric`, and all associated fields. `r` is the Reuters session object.

`deleteric(r,ric,fid)` deletes the fields specified by `fid` for the `ric`.

## Examples

Delete `myric` and all of its fields:

```
deleteric(r,'myric')
```

---

Delete the fields `fid1` and `fid2` from `myric`:

```
deleteric(r,'myric',{'fid1','fid2'})
```

## See Also

`reuters` | `addric` | `contrib` | `fetch`

## Purpose

Request data from Reuters data servers

## Syntax

```
d = fetch(r,s)
d = fetch(r,s,callback)
d = fetch(r,s,[],f)
```

## Arguments

| | |
|---|---|
| `r` | Reuters session object created with `reuters`. |
| `s` | Reuters security object. |
| `callback` | MATLAB function that runs for each data event that occurs. |

## Description

`d = fetch(r,s)` returns the current data for the security `s`, given the Reuters session object `r`.

`d = fetch(r,s,callback)` uses the Reuters session object `r` to subscribe to the security `s`. MATLAB runs the `callback` function for each data event that occurs.

`d = fetch(r,s,[],f)` requests the given fields `f`, for the security `s`, given the Reuters session object `r`.

## Examples

### Retrieving Current Securities Data

Retrieve the current data for the security `GOOG.O` using the Reuters session object `r`:

```
d = fetch(r,'GOOG.O')
```

Following is a partial listing of the returned security data:

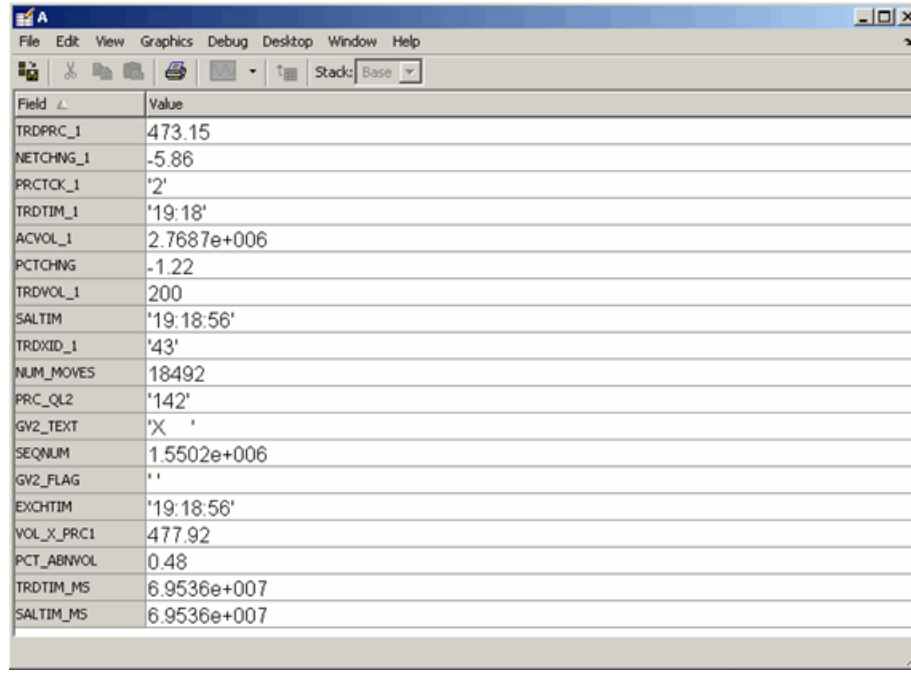
```
d =
PROD_PERM: 74.00
```

```
RDNDISPLAY: 66.00
DSPLY_NAME: 'DELAYED-15GOOGLE'
RDN_EXCHID: 'O'
TRDPRC_1: 474.28
TRDPRC_2: 474.26
TRDPRC_3: 474.25
TRDPRC_4: 474.25
TRDPRC_5: 474.25
NETCHNG_1: -4.73
HIGH_1: 481.35
LOW_1: 472.78
PRCTCK_1: '1'
CURRENCY: '840'
TRADE_DATE: '30 APR 2007
```

### Subscribing to a Security

To subscribe to a security and process the data in real time, specify a `callback` function. MATLAB runs this function each time it receives a real-time data event from Reuters. In this example, the callback function, `rtdemo`, returns the subscription handle associated with this request to the base MATLAB workspace, `A`. The `openvar` function is then called to display `A` in the Variables editor. A partial list of the data included in `A` appears in the figure.

```
d = fetch(r,'GOOG.O','rtdemo')
openvar('A')
```




| Field | Value |
|---|---|
| TRDPRC_1 | 473.15 |
| NETCHNG_1 | -5.86 |
| PRCTCK_1 | '2' |
| TRDTIM_1 | '19:18' |
| ACVOL_1 | 2.7687e+006 |
| PCTCHNG | -1.22 |
| TRDVOL_1 | 200 |
| SALTIM | '19:18:56' |
| TRDXID_1 | '43' |
| NUM_MOVES | 18492 |
| PRC_QL2 | '142' |
| GV2_TEXT | 'X ' |
| SEQNUM | 1.5502e+006 |
| GV2_FLAG | ' ' |
| EXCHTIM | '19:18:56' |
| VOL_X_PRC1 | 477.92 |
| PCT_ABNVOL | 0.48 |
| TRDTIM_MS | 6.9536e+007 |
| SALTIM_MS | 6.9536e+007 |

## See Also

`reuters` | `close` | `stop`

# get

## Purpose

Retrieve properties of Reuters session objects

## Syntax

```
e = get(r)
e = get(r,f)
```

## Arguments

| | |
|---|---|
| `r` | Reuters session object created with `reuters`. |
| `f` | Reuters session properties list. |

## Description

`e = get(r)` returns Reuters session properties for the Reuters session object `r`.

`e = get(r,f)` returns Reuters session properties specified by the properties list `f` for the Reuters session object `r`.

## See Also

`reuters`

## Purpose

Request data from Reuters Time Series One

## Syntax

```
d = history(r,s)
d = history(r,s,p)
d = history(r,s,f)
d = history(r,s,f,p)
d = history(r,s,d)
d = history(r,s,startdate,enddate)
d = history(r,s,startdate,enddate,p)
d = history(r,s,f,startdate,enddate)
d = history(r,s,f,startdate,enddate,p)
```

## Description

`d = history(r,s)` returns all available daily historical data for the RIC, `s`, for the Reuters session object `r`.

`d = history(r,s,p)` returns all available historical data for the RIC, `s`, for the Reuters session object `r`. `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(r,s,f)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `r`.

`d = history(r,s,f,p)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `r`. `p` specifies the period of the data.

`d = history(r,s,d)` returns the historical data for the RIC, `s`, for the given date, `d`, for the Reuters session object `r`.

`d = history(r,s,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(r,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(r,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(r,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(r,'WXYZ.O')` returns a structure containing all available historical end of day daily data for the RIC `wxyz.o`, for the Reuters session object `r`.

`d = history(r,'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(r,'WXYZ.O','close','m')` returns all available monthly data.

`d = history(r,'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(r,'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(r,'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

`reuters` | `fetch`

# stop

## Purpose

Unsubscribe securities

## Syntax

```
stop(r)
stop(r,d)
```

## Arguments

| | |
|---|---|
| `r` | Reuters session object created with `reuters`. |
| `d` | Subscription handle returned by `fetch`. |

## Description

`stop(r)` unsubscribes all securities associated with the Reuters session object `r`.

`stop(r,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 `r`:

```
stop(r,d)
```

Unsubscribe all securities associated with the Reuters connection object `r`:

```
stop(r)
```

## See Also

`reuters | fetch`

## Purpose

Reuters Market Data System configuration editor

## Syntax

```
rmdsconfig
```

## Description

`rmdsconfig` opens the Reuters Market Data System configuration editor.

## See Also

`reuters`

# rnseloader

## Purpose

Retrieve data from Reuters Newscope 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` | Reuters Newscope 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 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 Reuters Newscope 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

`reuters` | `rdthloader`

## Purpose

SIX Financial Information connection

## Syntax

```
T = tlkrs(CI,UI,password)
```

## Description

`T = tlkrs(CI,UI,password)` makes a connection to the SIX Financial Information data service given the Customer ID (`CI`), User ID (`UI`), and password (`password`) provided by SIX Financial Information.

## See Also

`close` | `getdata` | `history` | `timeseries`

# close

**Purpose** Close connection to SIX Financial Information

**Syntax** `close(C)`

**Description** `close(C)` closes the connection, `C`, to SIX Financial Information.

**See Also** `tlkrs`

## Purpose

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

`tlkrs` | `history` | `timeseries` | `tkfieldtoid` | `tkidtofield`

## Purpose

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 strings in `d.P.k` to their corresponding field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

## See Also

`tlkrs` | `getdata` | `timeseries` | `tkfieldtoid` | `tkidtofield`

## Purpose

True if valid SIX Financial Information connection

## Syntax

```
X = isconnection(C)
```

## Description

`X = isconnection(C)` returns `true` if `C` is a valid SIX Financial Information connection and `false` otherwise.

## See Also

`tlkrs` | `close` | `getdata`

# timeseries

## Purpose

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, and 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

`tlkrs` | `getdata` | `history`

## Purpose

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

`tlkrs` | `getdata` | `history` | `timeseries` | `tkidtofield`

# tkidtofield

## Purpose

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 `tkidtofield`:

```
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

`tlkrs` | `getdata` | `history` | `timeseries` | `tkfieldtoid`

# yahoo

## Purpose

Connect to Yahoo! Finance

## Syntax

```
c = yahoo
```

## Description

`c = yahoo` verifies that the URL `http://download.finance.yahoo.com` is accessible and creates a Yahoo! connection object.

## Output Arguments

### c - Yahoo! connection

connection object

Yahoo! connection, returned as a connection object.

## Examples

### Connect to Yahoo! Finance

```
c = yahoo
```

```
c =

  yahoo with properties:

     url: 'http://download.finance.yahoo.com'
      ip: []
    port: []
```

`yahoo` returns a successful connection `c` with empty `ip` and `port` properties.

Close Yahoo! connection.

```
close(c);
```

## See Also

`builduniverse` | `close` | `fetch` | `get` | `isconnection` | `trpdata`

## Purpose

Portfolio matrix with total return price data from Yahoo!

## Syntax

```
X = builduniverse(y,s,d1,d2,p)
```

## Description

`X = builduniverse(y,s,d1,d2,p)` builds a portfolio matrix using Yahoo! data to compute a total return price series. `X` is an $m$-by-($n$ + 1) matrix, where $m$ refers to the number of records of data and $n$ refers to the number of securities. Column 1 of the matrix contains MATLAB date numbers and the remaining columns are the total return prices for each security. `y` is the Yahoo! connection handle, `s` is a cell array of security identifiers, `d1` and `d2` are the start and end dates for the data request, and `p` is the periodicity flag. `p` can be entered as:

- `'d'` for daily values
- `'w'` for weekly values
- `'m'` for monthly values

## Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by the `builduniverse` function contains the expected results.

## Examples

Compute a total return price series and convert to daily total returns:

```
y = yahoo;
%
% Load security list.
s = {'A', 'B', 'C'};

% Get a daily total return price series for securities.
% (Calculated from prices, splits and dividends.)
Universe = builduniverse(y,s,'1/15/2007',floor(now));

% Convert to daily total returns.
Universe = periodicreturns(Universe,'d');
```

## See Also

`fetch` | `trpdata`

# close

## Purpose

Close connections to Yahoo! Finance

## Syntax

```
close(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Yahoo! connection object created with `yahoo`. |

## Description

`close(Connect)` closes the connection to the Yahoo! Finance.

## See Also

`yahoo`

## Purpose

Request data from Yahoo! Finance

## Syntax

```
d = fetch(c,s)
d = fetch(c,s,date)
d = fetch(c,s,fromdate,todate)
d = fetch(...,period)

d = fetch(c,s,f)
d = fetch(c,s,f,date)
d = fetch(c,s,f,fromdate,todate)
d = fetch(...,period)
```

## Description

`d = fetch(c,s)` returns data for all fields from Yahoo! web site for the indicated security.

---

**Note** This function does not support retrieving multiple securities at once. You must fetch a single security at a time.

---

`d = fetch(c,s,date)` returns all security data for the requested date.

`d = fetch(c,s,fromdate,todate)` returns security data for the date range `fromdate` through `todate`.

`d = fetch(...,period)` returns security data with the indicated period.

`d = fetch(c,s,f)` returns data for the specified fields.

`d = fetch(c,s,f,date)` returns security data on the requested date.

`d = fetch(c,s,f,fromdate,todate)` returns security data for the date range `fromdate` through `todate`.

`d = fetch(...,period)` returns security data with the indicated period.

## Input Arguments

### c - Yahoo! connection

connection object

Yahoo! connection, specified as a connection object created using `yahoo`.

### s - Security list

string | cell array

Security list, specified as a string for one security or a cell array of strings for more than one security. Security strings must be in a format recognizable by the Yahoo! server.

---

**Note** Retrieving historical data for multiple securities at one time is not supported for Yahoo!. You can fetch historical data for a single security at a time.

---

**Data Types**
`char | cell`

### date - Request date

string | serial date number

Request date, specified as a string or a serial date number indicating the date for the requested data. If you enter today's date, `fetch` returns yesterday's data.

**Data Types**
`double | char`

### fromdate - Beginning date

scalar | vector | matrix | string | cell array

Beginning date for the historical data, specified as a double scalar, double vector, double matrix, string, or cell array of strings. You can

specify dates in any format supported by `datestr` and `datenum` that show a year, month, and day.

**Data Types**
`double | char | cell`

### todate - End date
scalar | vector | matrix | string | cell array

End date for the historical data, specified as a double scalar, double vector, double matrix, string, or cell array of strings. You can specify dates in any format supported by `datestr` and `datenum` that show a year, month, and day.

**Data Types**
`double | char | cell`

### period - Period
string

Period within a date range, specified as a string. Possible values are:

- `'d'`: daily
- `'w'`: weekly
- `'m'`: monthly
- `'v'`: dividends

**Data Types**
`char`

### f - Request fields
string | cell array

Request fields, specified as a string or cell array of strings indicating the data fields for which to retrieve data. A partial list of supported values for current market data are:

- `'Symbol'`
- `'Last'`

- 'Date'
- 'Time'

**Note** 'Date' and 'Time' are MATLAB date numbers. ('Time' is a fractional part of a date number. For example, 0.5 = 12:00:00 PM.)

- 'Change'
- 'Open'
- 'High'
- 'Low'
- 'Volume'

A partial list of supported values for historical data are:

- 'Close'
- 'Date'
- 'High'
- 'Low'
- 'Open'
- 'Volume'
- 'Adj Close'

For a complete list of supported values for market and historical data, see yhfields.mat.

**Data Types**
char | cell

## Output Arguments

**d - Output data**
structure | matrix

Output data, returned as a structure or double matrix containing the requested data retrieved from Yahoo! Finance.

## Examples

### Retrieve Data for a Single Security

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the security data for IBM.

```
d = fetch(c,'IBM')
```

```
d =

    Symbol: {'IBM'}
      Last: 173.84
      Date: 735529.00
      Time: 0.42
    Change: 0.98
      Open: 173.23
      High: 173.84
       Low: 172.95
    Volume: 1132526.00
```

`fetch` returns a structure with the security name, last price, date, time, change, open price, high price, low price, and volume.

Close Yahoo! connection.

```
close(c);
```

### Retrieve Data on a Specified Date

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the security data for IBM with today's date.

```
d = fetch(c,'IBM',now)
```

```
d =

     735528.00         174.42         174.75         172.63         172.86
```

`fetch` returns the date, open price, high price, low price, closing price, volume, and adjusted close price.

Close Yahoo! connection.

```
close(c);
```

### Retrieve the Last Prices for a Set of Equities

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the last prices for the `ko`, `pep`, and `mcd` equities.

```
FastFood = fetch(c,{'ko', 'pep', 'mcd'},'Last')
```

```
FastFood =
    Last: [3x1 double]
```

`fetch` returns a structure with the last prices.

Display the last prices.

```
FastFood.Last
```

```
ans =
         42.96
         45.71
         23.70
```

Close Yahoo! connection.

```
close(c);
```

### Retrieve a Closing Price on a Specified Date

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the closing price for the `ko` equity on April 6, 2010.

```
ClosePrice = fetch(c,'ko','Close','Apr 6 2010')
```

```
ClosePrice =

     734234.00         54.29
```

`fetch` returns the date in the first column and the closing price in the second column.

Close Yahoo! connection.

```
close(c);
```

### Retrieve a Closing Price with a Date Range

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the closing price for IBM from August 1, 1999 to August 25, 1999.

```
ClosePrice = fetch(c,'IBM','Close','08/01/99','08/25/99')
```

```
ClosePrice =

     730357.00        122.37
     730356.00        122.00
     730355.00        124.44
     730352.00        121.75
     730351.00        122.94
```

```
    ...
```

`fetch` returns the date in the first column and the closing price in the second column.

Close Yahoo! connection.

```
close(c);
```

### Retrieve a Security Data with a Date Range

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain data for IBM from February 1, 2000 through February 20, 2000.

```
d = fetch(c,'IBM','2/1/2000','2/20/2000')
```

```
d =

    730534.00       115.25       115.94       111.50       112.50
    730533.00       116.50       118.87       115.75       116.75
    730532.00       116.50       117.31       115.25       115.75
    730531.00       115.87       117.44       113.87       117.12
    730530.00       116.00       116.37       114.50       116.06
    ...
```

`fetch` returns the date, open price, high price, low price, closing price, volume, and adjusted close price in the columns. A row contains data for each trading day.

Close Yahoo! connection.

```
close(c);
```

### Retrieve the Daily Volume

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the daily volume for IBM for the last 300 days.

```
d = fetch(c,'IBM','Volume',now-300,now-1,'d')
```

```
d =

     735528.00    7079500.00
     735525.00   10548000.00
     735524.00   22358300.00
     735523.00    6615300.00
     735522.00    3365100.00
     ...
```

`fetch` returns the date in the first column and the volume in the second column.

Close Yahoo! connection.

```
close(c);
```

### Retrieve Stock Dividend Data

Connect to Yahoo! Finance.

```
c = yahoo;
```

Obtain the cash dividend data for IBM for the last 300 days.

```
d = fetch(c,'IBM',now-300,now-1,'v')
```

```
d =

     735453.00          0.95
     735362.00          0.95
     735271.00          0.85
```

`fetch` returns the date in the first column and cash dividend in the second column.

Close Yahoo! connection.

```
close(c);
```

## See Also

close | get | isconnection | yahoo

## Purpose

Retrieve properties of Yahoo! connection objects

## Syntax

```
value = get(Connect, 'PropertyName')
value = get(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Yahoo! connection object created with `yahoo`. |
| `PropertyName` | (Optional) A MATLAB string or cell array of strings containing property names. Currently the only property name recognized is `'url'`. |

## Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the Yahoo! 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.

## Examples

Connect to a Yahoo! Finance:

```
c = yahoo
c =

     url: 'http://download.finance.yahoo.com'
      ip: []
    port: []
```

Retrieve the URL of the connection:

```
get(c, 'url')

ans =

http://download.finance.yahoo.com
```

## See Also

`close` | `fetch` | `isconnection` | `yahoo`

# isconnection

## Purpose

Verify whether connections to Yahoo! Finance are valid

## Syntax

```
x = isconnection(Connect)
```

## Arguments

| | |
|---|---|
| `Connect` | Yahoo! connection object created with `yahoo`. |

## Description

`x = isconnection(Connect)` returns `x = 1` if the connection is a valid Yahoo! connection, and `x = 0` otherwise.

## Examples

Connect to a Yahoo! Finance:

```
c = yahoo
```

Verify that the connection, `c`, is valid:

```
x = isconnection(c)
x = 1
```

## See Also

`close` | `fetch` | `get` | `yahoo`

## Purpose

Total return price series data

## Syntax

```
[prc,act,div] = trpdata(y,s,d1,d2,p)
```

## Description

`[prc,act,div] = trpdata(y,s,d1,d2,p)`, where `y` is the Yahoo! connection handle, `s` is the security string, `d1` is the start date, `d2` is the end date, and `p` is the periodicity flag for Yahoo!, generates a total return price series. `prc` is the price, `act` is the action, and `div` is the dividend returned in the total return price series.

## Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by the `trpdata` function contains the expected results.