Datafeed Toolbox™

User's Guide

MATLAB®

R2015a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Datafeed Toolbox™ User's Guide

Revision History

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

Contents

1 Getting Started

2 Communicate with Financial Data Servers

3 Data Provider Workflows

Getting Started

Datafeed Toolbox Product Description

Access financial data from data service providers

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

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

Supported data providers include Bloomberg®, FactSet®, FRED®, Haver Analytics®, Interactive Data™, IQFEED®, Kx Systems®, Inc., SIX Financial Information, Thomson Reuters™, and Yahoo!® Finance.

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
- Bloomberg Desktop, B-PIPE®, and Server connectivity
- Thomson Reuters Eikon® Desktop, RMDS, Datastream®, NewsScope, and Tick History connectivity
- Connectivity to FactSet, Interactive Data, IQFEED, Kx Systems, SIX Financial Information, Yahoo! Finance, and other financial data providers
- Haver Analytics and Federal Reserve Economic Data (FRED) economic data support

Data Server Connection Requirements

In this section...

License Requirements

You must have a valid license for the required client software on your machine. For details, contact your data service sales representative or go to the data service provider website. For the list of websites, see "Communicate with Data Providers" on page 2-2. These data service providers require you to install proprietary software on your computer.

- Bloomberg
  - A Bloomberg Desktop, Server, or B-PIPE software license.
  - A license for Bloomberg Data License.
- Interactive Data
  - A license to use Interactive Data RemotePlus^{SM}.
- Haver Analytics
- Kx Systems, Inc.
- IQFEED
- FactSet
  - A license to use FactSet DataDirect or FactSet Workstation.
- Thomson Reuters data servers:
  - A license for Thomson Reuters Datastream DataWorks®.
  - To connect from the Internet to the Thomson Reuters Datastream API, request a user name, password, and URL from Thomson Reuters.
  - A license for Thomson Reuters Eikon.

Proxy Information Requirements

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

- FactSet
- FRED
- Thomson Reuters Eikon
- Thomson Reuters Datastream
- Thomson Reuters Tick History
- Yahoo!

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

Bloomberg Connection Requirements

To connect to Bloomberg, install specific Java® archive (JAR) files and add them to the MATLAB Java class path. For information about your specific Bloomberg connection, refer to one of the following sections.

Once you install the JAR files, add these files to the MATLAB Java class path for every MATLAB session using `javaaddpath`. To automate adding these files, add `javaaddpath` to your `startup.m` file or add the full path for the JAR file to your `javaclasspath.txt` file. To decide which way is best for you, see “Startup Options in MATLAB Startup File” and “Static Path”.

Bloomberg Desktop, Server, and B-PIPE

With the Bloomberg V3 release, install the JAR file `blpapi3.jar` from Bloomberg to connect to Bloomberg Desktop, Server, and B-PIPE. This JAR file ensures that `blp`, `blpsrv`, `bpipe`, and other Bloomberg commands work correctly.

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

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

1 In your Bloomberg terminal, type `WAPI {GO}` to open the API Developer's Help Site screen.
2 Click API Download Center, then click Desktop API.
3 Once you have `blpapi3.jar` on your system, add it to the MATLAB Java class path using the preceding JAR file instructions.

Bloomberg Data License

To connect to Bloomberg Data License, add these JAR files to the MATLAB Java class path using the preceding JAR file instructions.

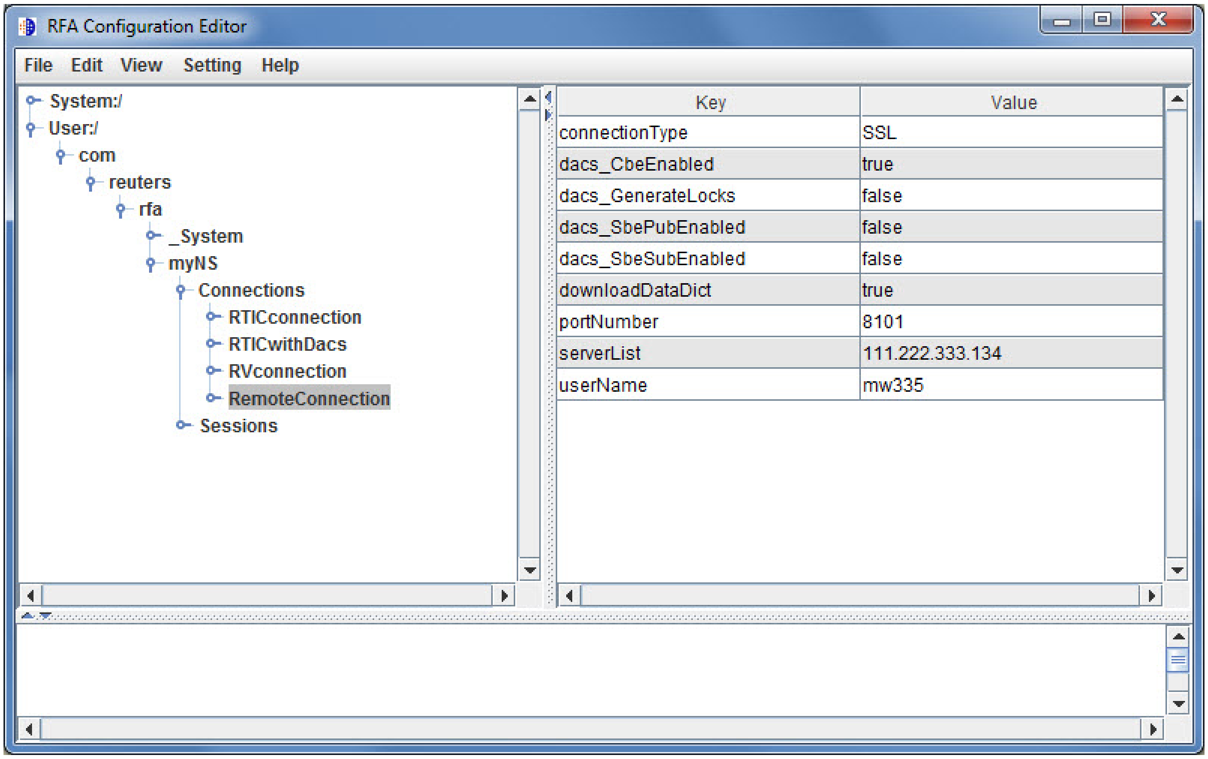
- `bbdl.jar`
- `bbdlftp.jar`
- `bbdlapi.jar`

For details about these JAR files, see the Data License Java SE API folder. Find this folder by entering `DLSD` and clicking **<GO>** in the Bloomberg terminal.

Reuters Configurations

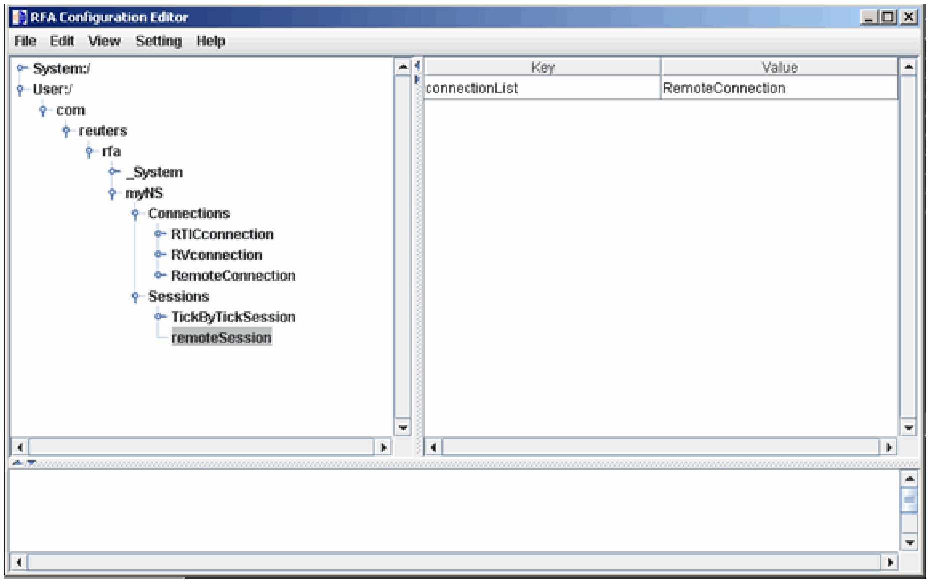
Configure a Reuters Connection

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



Key	Value
connectionType	SSL
dacs_CbeEnabled	true
dacs_GenerateLocks	false
dacs_SbePubEnabled	false
dacs_SbeSubEnabled	false
downloadDataDict	true
portNumber	8101
serverList	111.222.333.134
userName	mw335

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



Key	Value
connectionList	RemoteConnection

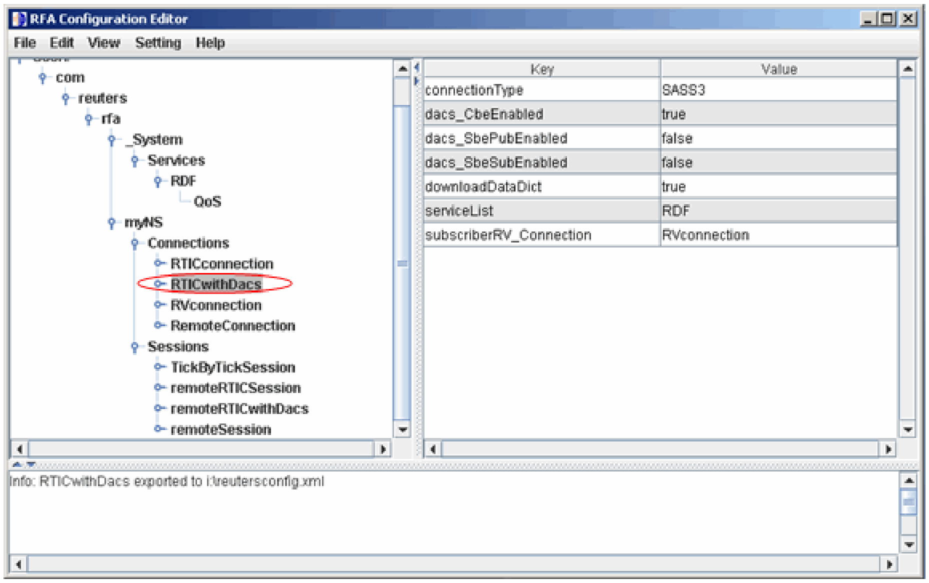
5 To connect without Data Access Control System (DACS) authentication, disable DACS by setting the keys in `RemoteConnection` to the values as shown in this table.

Key	Value
`dacs_CbeEnabled`	`false`
`dacs_SbePubEnabled`	`false`
`dacs_SbeSubEnabled`	`false`

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

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

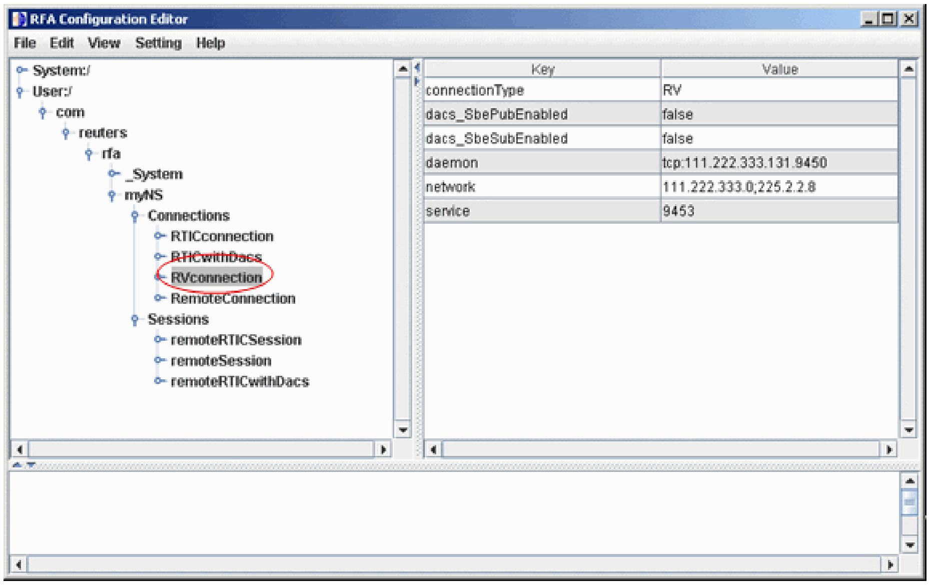
1 After loading and modifying the sample configuration file, set the keys and value fields as shown in the RFA Configuration Editor for the connection `RTICwithDacs`.



Key	Value
connectionType	SASS3
dacs_CbeEnabled	true
dacs_SbePubEnabled	false
dacs_SbeSubEnabled	false
downloadDataDict	true
serviceList	RDF
subscriberRV_Connection	RVconnection

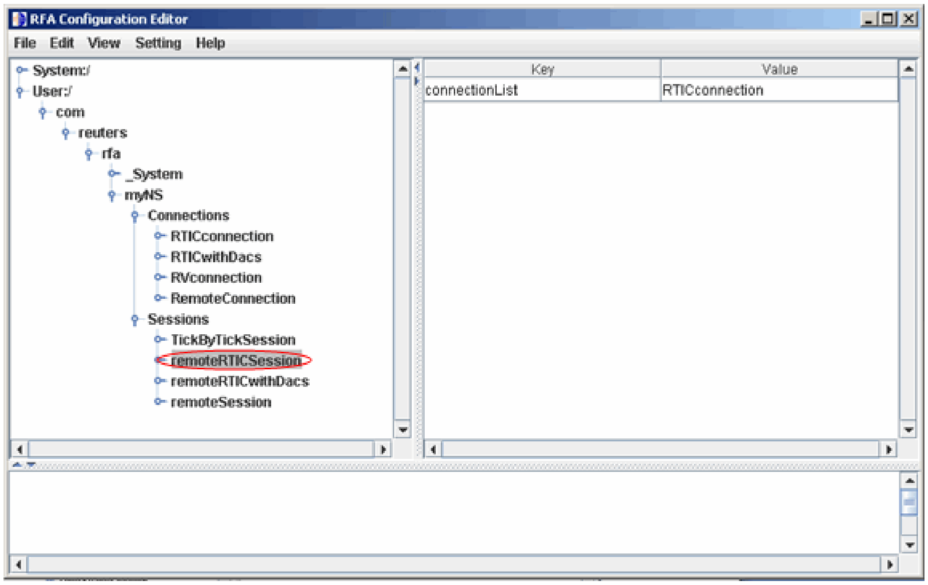
```
Info: RTICwithDacs exported to i:\reutersconfig.xml
```

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



Key	Value
connectionType	RV
dacs_SbePubEnabled	false
dacs_SbeSubEnabled	false
daemon	tcp:111.222.333.131.9450
network	111.222.333.0;225.2.2.8
service	9453

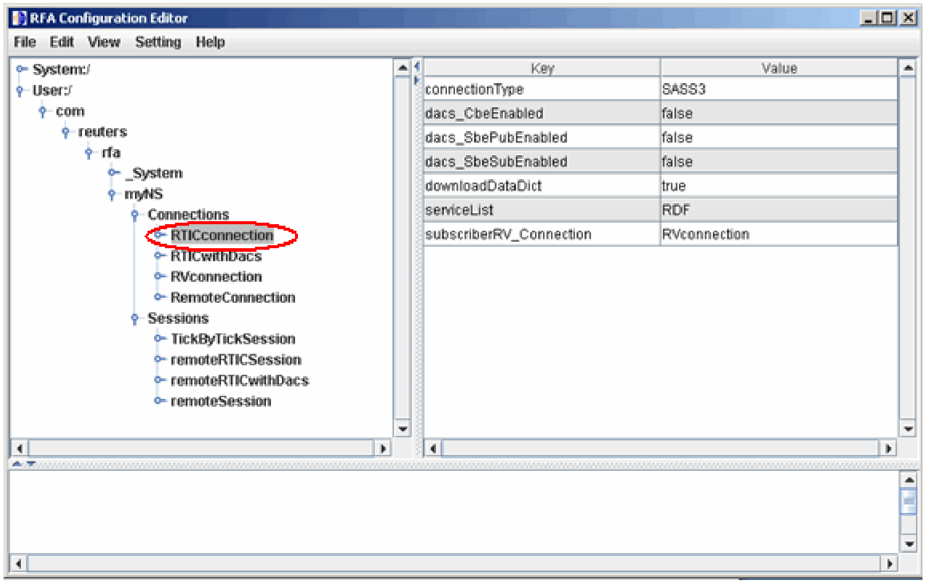
The RFA Configuration Editor shows the session `remoteRTICSession` referencing the `RTICConnection` connection.



Key	Value
connectionList	RTICconnection

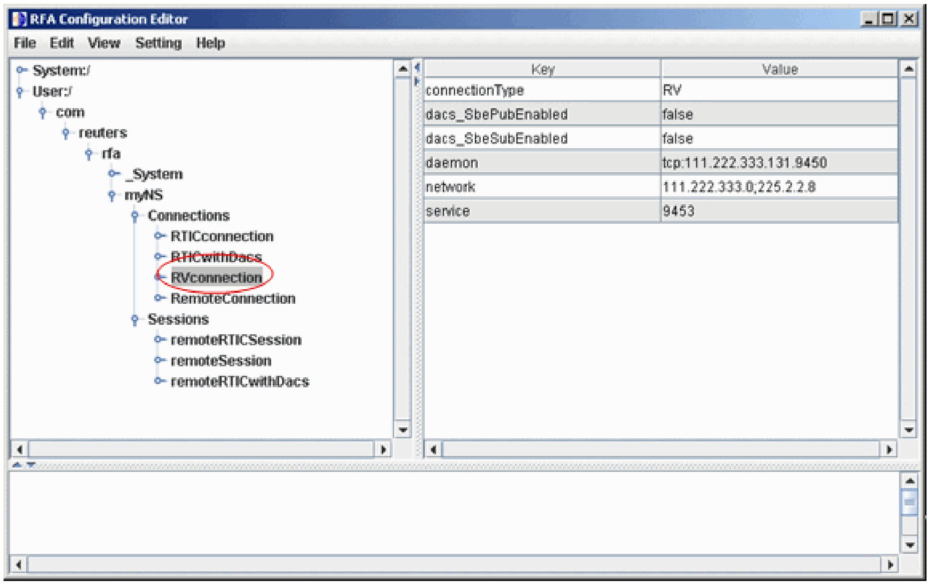
Configure an RTIC (TIC-RMDS Edition) Reuters Connection Without DACS Authentication

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



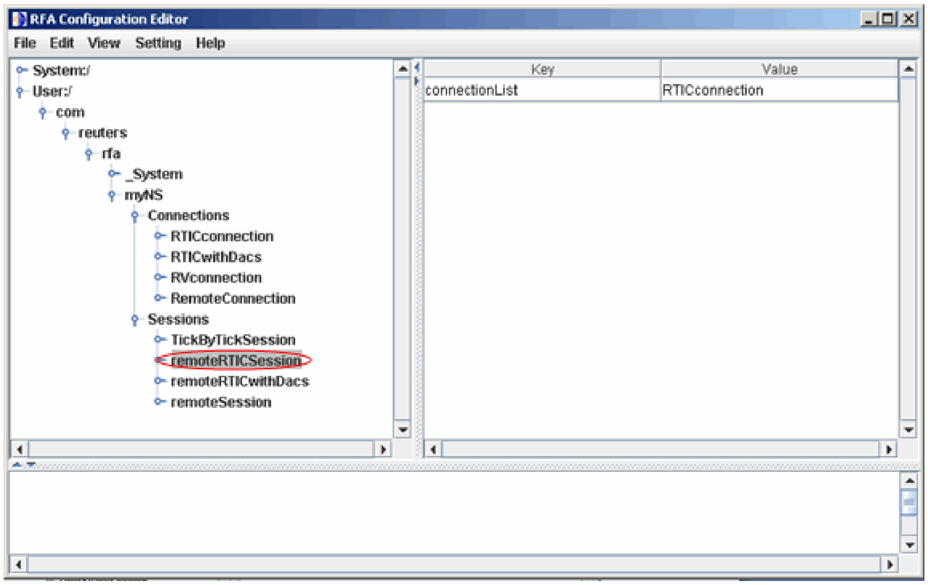
Key	Value
connectionType	SASS3
dacs_CbeEnabled	false
dacs_SbePubEnabled	false
dacs_SbeSubEnabled	false
downloadDataDict	true
serviceList	RDF
subscriberRV_Connection	RVconnection

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



Key	Value
connectionType	RV
dacs_SbePubEnabled	false
dacs_SbeSubEnabled	false
daemon	tcp:111.222.333.131.9450
network	111.222.333.0;225.2.2.8
service	9453

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



Key	Value
connectionList	RTICconnection

Troubleshoot the Reuters Configuration Editor

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

```
java com.reuters.rfa.tools.config.editor.ConfigEditor
org.xml.sax.SAXException: System property
org.xml.sax.driver not specified
at org.xml.sax.helpers.XMLReaderFactory.createXMLReader(Unknown
Source)
at com.reuters.rfa.tools.config.editor.rfaConfigRuleDB.rfaConfi
gRuleDB.java:56)
at com.reuters.rfa.tools.config.editor.ConfigEditor.init
(ConfigEditor.java:86)
at (com.reuters.rfa.tools.config.editor.ConfigEditor.
(ConfigEditor.java:61) at
com.reuters.rfa.tools.config.editor.ConfigEditor.main
(ConfigEditor.java:1303)
```

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

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

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

- When you establish a connection with DACS authentication, if these messages or similar messages appear in the Command Window:

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

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

Retrieving Current and Historical Data Using Bloomberg

This example shows how to connect to Bloomberg and retrieve current and historical Bloomberg market data.

Connect to Bloomberg

```
c = blp;
```

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

Retrieve Current Data

Retrieve closing and open prices for Microsoft®.

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

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

```
d =

    LAST_PRICE: 36.95
          OPEN: 36.94

sec =

    MSFT US Equity
```

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

Retrieve Historical Data

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

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

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

```
d =

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

sec =

    MSFT US Equity
```

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

Close the Bloomberg Connection

```
close(c)
```

See Also

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

Retrieving Current and Historical Data Using Thomson Reuters

This example shows how to connect to the Reuters Market Data System (RMDS) and retrieve current and historical Thomson Reuters market data.

Connect to Thomson Reuters

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

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

Retrieve Current Data

Retrieve current data for Google®.

```
sec = 'GOOG.O';

d = fetch(c,sec)

d = 

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

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

Close the Thomson Reuters connection.

```
close(c)
```

Retrieve Historical Data

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

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

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

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

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

```
d = 

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

`d` is a structure with the following fields:

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

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

Display the open price.

```
d.OPEN
```

```
ans =

        702.24
```

```
          679.50
          759.05
          ...
```

Close the Thomson Reuters Connection

```
close(c)
```

See Also

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

Retrieving Historical Data Using FRED

This example shows how to connect to FRED and retrieve historical data.

Connect to FRED

```
c = fred;
```

Retrieve All Historical Data

Retrieve all historical data for the U.S. / Euro Foreign Exchange Rate series.

```
series = 'DEXUSEU';

d = fetch(c,series)
```

```
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 Seasonally Adjusted'
             Frequency: ' Daily'
                 Units: ' U.S. Dollars to One Euro'
             DateRange: ' 1999-01-04 to 2013-12-13'
           LastUpdated: ' 2013-12-16 4:51 PM CST'
                 Notes: ' Noon buying rates in New York City for cable transfers payable in foreign currencies.'
                  Data: [3900x2 double]
```

`d` contains the series description.

Display the numeric representation of the date and the foreign exchange rate.

```
d.Data
```

```
ans =

     730124.00           1.18
     730125.00           1.18
     730126.00           1.16
     ...
```

Retrieve Historical Data for a Date Range

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

```
fromdate = '01/01/2012'; % beginning of date range for historical data
```

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

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

Close the FRED Connection

```
close(c)
```

See Also

close | fetch | fred

Retrieving Historical Data Using Haver Analytics

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

Connect to Haver Analytics

Connect to Haver Analytics using a daily file.

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

Retrieve All Historical Data

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

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

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

Display the first three rows of data.

```
d(1:3,:)

ans =

     715511.00           2.38
     715512.00           2.50
     715515.00           2.50
```

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

Retrieve Historical Data for a Date Range

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

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

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

Close the Haver Analytics Connection

```
close(c)
```

Open the Haver Analytics User Interface

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

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

```
havertool(c)
```

For details, see the `havertool` function.

See Also

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

Retrieving Intraday and Historical Data Using IQFEED

This example shows how to connect to IQFEED and retrieve intraday and historical data.

Connect to IQFEED

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

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

Retrieve Intraday Data

Retrieve today's intraday data for IBM®.

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

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

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

Display the first three rows of intraday data.

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

```
ans =

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

The columns in `IQFeedTimeseriesData` are:

- Timestamp.
- Last price.
- Last size.
- Total volume.
- Bid price.
- Ask price.

- Tick identifier.
- The last column is the basis for last trade.

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Historical Data

Connect to IQFEED.

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

Retrieve the last five weeks of historical data for IBM.

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

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

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

Display the first three rows of historical weekly data.

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

```
ans =

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

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

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

- Volume
- Open interest

Close the IQFEED Connection

```
close(c)
```

See Also

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

Retrieving Current and Historical Data Using Yahoo!

This example shows how to connect to Yahoo! and retrieve current and historical data.

Connect to Yahoo!

```
c = yahoo;
```

Retrieve Current Data

Retrieve data for IBM on January 3, 2014.

```
sec = 'IBM';
sdate = '01/03/2014'; % retrieve data for a single date

d = fetch(c,sec,sdate)
```

```
d =

    735584.00        173.22        178.35        172.73        177.85    7517000.00        177.85
```

`d` contains the numeric representation of the date, open price, high price, low price, closing price, volume, and adjusted closing price. `sec` contains the Yahoo! security name for IBM.

Retrieve Historical Data

Retrieve the closing prices from January 1, 2012 through June 30, 2012 for IBM.

```
field = 'Close'; % retrieve closing price data
fromdate = '01/01/2012'; % beginning of date range for historical data
todate = '06/30/2012'; % ending of date range for historical data

d = fetch(c,sec,field,fromdate,todate);
```

Display the first three rows of data.

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

```
ans =

     735049.00        195.58
     735048.00        191.40
     735047.00        193.00
```

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

Close the Yahoo! Connection

```
close(c)
```

See Also

`close` | `fetch` | `yahoo`

Writing and Running Custom Event Handler Functions

In this section...

Write a Custom Event Handler Function

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

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

For details, see "Create Functions in Files". For a code example of a Bloomberg event handler function, see `v3stockticker`.

Run a Custom Event Handler Function

You can run the custom event handler function by passing the function name as an input argument into an existing function. For Thomson Reuters RMDS function `fetch`, specify the custom event handler as a string. For other functions, specify the custom event handler function name either as a string or function handle. For details about function handles, see "What Is a Function Handle?"

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

Use a string.

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

Or, use a function handle.

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

Workflow for Custom Event Handler Function

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

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

See Also

`fetch` | `realtime`

More About

- “Create Functions in Files”
- “What Is a Function Handle?”

Communicate with Financial Data Servers

Communicate with Data Providers

Datafeed Toolbox supports connection to these data providers. This table lists the connection functions for each data provider. To communicate with your data service provider, start with these functions.

Data Provider	Website	Function
Bloomberg	http://www.bloomberg.com	`blp`, `blpsrv`, `bpipe`, or `bdl`
FactSet	http://www.factset.com	`factset` or `fds`
FRED	http://research.stlouisfed.org/fred2/	`fred`
Haver Analytics	http://www.haver.com	`haver`
Interactive Data	http://www.interactivedata-prd.com/	`idc`
IQFEED	http://www.iqfeed.net/	`iqf`
Kx Systems, Inc.	http://www.kx.com	`kx`
SIX Financial Information	http://www.six-financial-information.com	`tlkrs`
Thomson Reuters	http://www.thomsonreuters.com/	`datastream`, `reuters`, `rdth`, or `treikon`
Yahoo!	http://finance.yahoo.com	`yahoo`

Comparing Bloomberg Connections

Datafeed Toolbox uses three different Bloomberg services to connect to Bloomberg. Use the information in this table to learn about the functions for establishing each connection and the data access functionality of each service.

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

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

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

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

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

Data Provider Workflows

Connect to Bloomberg

This example shows how to create a connection to Bloomberg using these Bloomberg services: Bloomberg Desktop, Bloomberg Server, B-PIPE, and Bloomberg Data License. For details about Bloomberg connection requirements, see “Data Server Connection Requirements” on page 1-3.

Create the Bloomberg Desktop Connection

```
c = blp
```

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

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

Validate the connection `c`.

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

```
v =

     1
```

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

Retrieve the Bloomberg Desktop connection properties.

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

```
v =

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

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

Close the Bloomberg Desktop connection.

```
close(c)
```

Create the Bloomberg Server Connection

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

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

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

c = blpsrv(uuid,serverip)

c = 

  blpsrv with properties:

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

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

Close the Bloomberg Server connection.

```
close(c)
```

Create the B-PIPE Connection

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

- The authentication is Windows® Authentication by setting `authorizationtype` to `'OS_LOGON'`.
- The application name is blank because you are not connecting to Bloomberg B-PIPE using an application.

- The IP address `serverip` for the machine running the Bloomberg B-PIPE process is `'111.11.11.112'`.
- The port number is `8194`.

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

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

```
c = 

  bpipe with properties:

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

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

Close the B-PIPE connection.

```
close(c)
```

Create the Bloomberg Data License Connection

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

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

```
username = 'dl338';
password = 'Lb=cYaZ';
```

```
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'pDyJaV';

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

c = 

bdl with properties:

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

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

Close the Bloomberg Data License connection.

```
close(c)
```

See Also

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

Related Examples

- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Bloomberg Intraday Tick Data” on page 3-10
- “Retrieve Bloomberg Real-Time Data” on page 3-12
- “Retrieve Data Using the Bloomberg Data License” on page 3-14

More About

- “Data Server Connection Requirements” on page 1-3
- “Comparing Bloomberg Connections” on page 2-3
- “Workflow for Bloomberg” on page 3-17

Retrieve Bloomberg Current Data

This example shows how to retrieve current data from Bloomberg.

Connect to Bloomberg.

```
c = blp;
```

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

Retrieve last and open prices for Microsoft.

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

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

```
d = 

    LAST_PRICE: 36.95
          OPEN: 36.94

sec = 

    MSFT US Equity
```

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

Close the Bloomberg connection.

```
close(c)
```

See Also

`blp` | `close` | `getdata`

Related Examples

- "Connect to Bloomberg" on page 3-2
- "Retrieve Bloomberg Historical Data" on page 3-8
- "Retrieve Bloomberg Intraday Tick Data" on page 3-10

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

More About

- “Workflow for Bloomberg” on page 3-17

Retrieve Bloomberg Historical Data

This example shows how to retrieve historical data from Bloomberg.

Connect to Bloomberg.

```
c = blp;
```

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

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

```
sec = 'MSFT US Equity';
fields = {'LAST_PRICE';'OPEN'}; % Retrieve data for closing and open prices
fromdate = '1/01/2012'; % Start of date range for historical data
todate = '12/31/2012'; % End of date range for historical data
period = 'monthly'; % Retrieve monthly data

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

```
d =

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

sec =

    MSFT US Equity
```

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

Close the Bloomberg connection.

```
close(c)
```

See Also

`blp` | `close` | `history`

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Intraday Tick Data” on page 3-10
- “Retrieve Bloomberg Real-Time Data” on page 3-12

More About

- “Workflow for Bloomberg” on page 3-17

Retrieve Bloomberg Intraday Tick Data

This example shows how to retrieve intraday tick data from Bloomberg.

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

```
ans =

  Columns 1 through 7

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

Column 8

   38902968.00
    8779374.00
    9626896.00
   ...
```

The columns in `d` contain the following:

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

The first row of data shows that on today's date the open price is $187.20, the high price is $187.60, the low price is $187.02, the closing price is $187.08, the volume of ticks 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)
```

See Also

`blp` | `close` | `timeseries`

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Bloomberg Real-Time Data” on page 3-12

More About

- “Workflow for Bloomberg” on page 3-17

Retrieve Bloomberg Real-Time Data

This example shows how to retrieve real-time data from Bloomberg. You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3stockticker` to return Bloomberg stock tick data.

Create the Bloomberg connection.

```
c = blp;
```

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

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

`v3stockticker` requires the input argument `f` of `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 last trade price and volume.

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

Close the Bloomberg connection.

```
close(c)
```

See Also

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

Related Examples

- "Connect to Bloomberg" on page 3-2
- "Retrieve Bloomberg Current Data" on page 3-6
- "Retrieve Bloomberg Historical Data" on page 3-8
- "Retrieve Bloomberg Intraday Tick Data" on page 3-10

More About

- "Workflow for Bloomberg" on page 3-17
- "Writing and Running Custom Event Handler Functions" on page 1-29

Retrieve Data Using the Bloomberg Data License

This example shows how to retrieve Bloomberg Data License data with a request file using a Bloomberg Data License connection. To access the code for this example, see `BloombergDataLicenseWorkflow.m`.

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

   - The Bloomberg Data License SFTP server login name is `'dl111'`.
   - The Bloomberg Data License SFTP server password is `'Pc=zXdA'`.
   - The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
   - The Bloomberg Data License SFTP port number is `30206`.
   - The decryption code is `'nAcLeZ'`.

   ```
   username = 'dl111';
   password = 'Pc=zXdA';
   hostname = 'dlsftp.bloomberg.com';
   portnumber = 30206;
   decrypt = 'nAcLeZ';

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

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

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

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

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

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

   ```
   s =

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

   The output file `getdataoutput.out` is available.

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

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

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

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

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

```
d = 

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

`d` is a structure with these fields:

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

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

6 Display the output file header information.

```
d.Header
```

```
ans = 

           RUNDATE: '20140924'
       PROGRAMFLAG: 'oneshot'
          FIRMNAME: 'dl111'
          FILETYPE: 'pc'
     REPLYFILENAME: 'getdataoutput.out'
    PRICING_SOURCE: 'BVAL'
     CLOSINGVALUES: 'yes'
             SECID: 'TICKER'
         YELLOWKEY: 'Equity'
       PROGRAMNAME: 'getdata'
       TIMESTARTED: 'Wed Sep 24 10:19:59 EDT 2014'
      TIMEFINISHED: 'Wed Sep 24 10:20:17 EDT 2014'
```

7. Close the Bloomberg Data License connection.

   ```
   close(c)
   ```

See Also

`bdl` | `close` | `dir`

Related Examples

- "Connect to Bloomberg" on page 3-2

More About

- "Workflow for Bloomberg" on page 3-17

Workflow for Bloomberg

In this section...

You can use Bloomberg to monitor market price information.

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

To request current, historical, intraday tick, and real-time data using Bloomberg Desktop, Bloomberg Server, and Bloomberg B-PIPE connections:

1. Connect to Bloomberg using `blp`, `blpsrv`, or `bpipe`.
2. Ensure a successful Bloomberg connection by using `isconnection`. Request properties of the connection objects using `get`.
3. Look up information about securities, curves, or government securities using `lookup`. Request Bloomberg field information using `category`, `fieldinfo`, or `fieldsearch`.
4. Request current data for a security using `getdata`. Request bulk data with header information using `getbulkdata`.
5. Request equity screening data using `eqs`.
6. Request historical data for a security using `history`.
7. Request historical technical analysis using `tahistory`.
8. Request intraday tick data for a security using `timeseries`.
9. Request real-time data for a security using `realtime`. Stop real-time data updates using `stop`.
10. Close the Bloomberg connection by using `close`.

Bloomberg Data License Service

To connect and retrieve data using Bloomberg Data License:

1. Connect to Bloomberg Data License using `bdl`.

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

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-6
- “Retrieve Bloomberg Historical Data” on page 3-8
- “Retrieve Bloomberg Intraday Tick Data” on page 3-10
- “Retrieve Bloomberg Real-Time Data” on page 3-12
- “Retrieve Data Using the Bloomberg Data License” on page 3-14

More About

- “Comparing Bloomberg Connections” on page 2-3

Retrieve Thomson Reuters Eikon Current Data

This example shows how to connect to Thomson Reuters Eikon and retrieve current data.

Connect to Thomson Reuters Eikon

Create a Thomson Reuters Eikon connection `c`.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon;
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
```

```
ans =

Disconnected

ans =

Succeed

ans =

Connected
```

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

Retrieve Current Data

Retrieve last price and bid price data for Google.

```
s = 'GOOG.O';
fields = {'LAST','BID'}; % Last price and bid price fields

d = getdata(c,s,fields)
```

```
ans =

GOOG.O

d =

    LAST: {[1119.77]}
     BID: {[1119.41]}
```

`getdata` returns `d` as a structure containing the field `LAST` with the last price $1119.77 and the field `BID` with the bid price $1119.41 for Google.

Close the Thomson Reuters Eikon Connection

To close the Thomson Reuters Eikon connection, exit MATLAB.

See Also

`getdata` | `treikon`

Related Examples

- “Retrieve Thomson Reuters Eikon Historical Data” on page 3-21
- “Retrieve Thomson Reuters Eikon Real-Time Data” on page 3-24

More About

- “Workflow for Thomson Reuters Eikon” on page 3-27

Retrieve Thomson Reuters Eikon Historical Data

This example shows how to connect to Thomson Reuters Eikon and retrieve historical data.

Connect to Thomson Reuters Eikon

Create a Thomson Reuters Eikon connection `c`.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon;
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
```

```
ans =

Disconnected

ans =

Succeed

ans =

Connected
```

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

Retrieve Historical Data

Retrieve the weekly open, high, low, and close prices for Apple. Retrieve data for the last 30 days.

```
s = 'AAPL.O';
fields = {'DATE','OPEN','HIGH','LOW','CLOSE'};
startdate = floor(now)-30; % Beginning of date range as of 30 days ago
enddate = floor(now); % End of date range as of today
period = 'W'; % Weekly periodicity

d = history(c,s,fields,startdate,enddate,period)

d = 

    '4/4/2014 12:00:0...'    [539.23]    [543.48]    [530.58]    [531.82]
    '3/28/2014 12:00:...'    [538.42]    [549.00]    [534.25]    [536.86]
    '3/21/2014 12:00:...'    [527.70]    [536.24]    [525.20]    [532.87]
    '3/14/2014 12:00:...'    [528.36]    [539.66]    [523.00]    [524.69]
```

`d` is a cell array that contains five columns:

- Date and time
- Open price
- High price
- Low price
- Close price

Each row represents one week of data. The total number of rows equals the number of weeks in the requested date range.

Close the Thomson Reuters Eikon Connection

To close the Thomson Reuters Eikon connection, exit MATLAB.

See Also

`history` | `treikon`

Related Examples

- "Retrieve Thomson Reuters Eikon Current Data" on page 3-19

- “Retrieve Thomson Reuters Eikon Real-Time Data” on page 3-24

More About

- “Workflow for Thomson Reuters Eikon” on page 3-27

Retrieve Thomson Reuters Eikon Real-Time Data

This example shows how to connect to Thomson Reuters Eikon, retrieve real-time data, stop real-time data retrieval, and resume real-time data retrieval.

Connect to Thomson Reuters Eikon

Create a Thomson Reuters Eikon connection `c`.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon;
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
```

```
ans =

Disconnected

ans =

Succeed

ans =

Connected
```

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

Retrieve Real-Time Data

Retrieve real-time data for the last price and bid price for Google. The sample event handler `trerealtimeeventhandler` retrieves the real-time data to put into the MATLAB variable `trRealtimeData` in the Workspace browser.

```
s = 'GOOG.O';
fields = {'LAST','BID'};

subs = realtime(c,s,fields,@(varargin)trerealtimeeventhandler(varargin{:}))
```

```
subs =

    AdxRtListCOMObj: [1x1 System.__ComObject]
       AdxRtListObj: [1x1 ThomsonReuters.Interop.RTX.AdxRtListClass]
              Items: {'GOOG.O'}
             Fields: {'LAST'  'BID'}
         UpdateMode: [1x1 ThomsonReuters.Interop.RTX.RT_RunMode]
```

`subs` is a subscription structure that contains the security list in the field `Items`. `subs` contains the Thomson Reuters Eikon field list in the structure field `Fields`.

Display the real-time data for Google by accessing the contents of the variable `trRealtimeData` in the Workspace browser.

```
trRealtimeData
```

```
trRealtimeData =

      RIC: 'GOOG.O'
     LAST: 561.26
      BID: 561.16
```

The variable `trRealtimeData` is a structure that contains real-time data. `trRealtimeData` contains the Thomson Reuters Eikon Reuters Instrument Code (RIC) in the structure field `RIC`. This structure contains any requested Thomson Reuters Eikon fields as structure fields. For example, `trRealtimeData` contains the last price of $561.26 for Google in the structure field `LAST`.

Stop Real-Time Data Retrieval

To stop real-time data retrieval, use the `stop` function with the subscription structure `subs`.

```
stop(c,subs)
```

Resume Real-Time Data Retrieval

To resume real-time data retrieval, use the `start` function with the subscription structure `subs`.

```
start(c,subs)
```

Close the Thomson Reuters Eikon Connection

To close the Thomson Reuters Eikon connection, exit MATLAB.

See Also

`realtime` | `start` | `stop` | `treikon`

Related Examples

- “Retrieve Thomson Reuters Eikon Current Data” on page 3-19
- “Retrieve Thomson Reuters Eikon Historical Data” on page 3-21

More About

- “Workflow for Thomson Reuters Eikon” on page 3-27
- “Writing and Running Custom Event Handler Functions” on page 1-29

Workflow for Thomson Reuters Eikon

You can use Thomson Reuters Eikon to monitor market price information.

To request current, historical, or real-time data:

1. Connect to Thomson Reuters Eikon using `treikon`.
2. Retrieve current data for a security using `getdata`.
3. Retrieve historical data for a security using `history`.
4. Retrieve real-time data for a security using `realtime`.
5. Start and stop real-time data updates using `start` and `stop`.
6. Retrieve chain data for a security using `chain`.
7. Close the Thomson Reuters Eikon connection by exiting MATLAB.

Related Examples

- "Retrieve Thomson Reuters Eikon Current Data" on page 3-19
- "Retrieve Thomson Reuters Eikon Historical Data" on page 3-21
- "Retrieve Thomson Reuters Eikon Real-Time Data" on page 3-24

Datafeed Toolbox Graphical User Interface

Retrieving Data Using the Datafeed Dialog Box

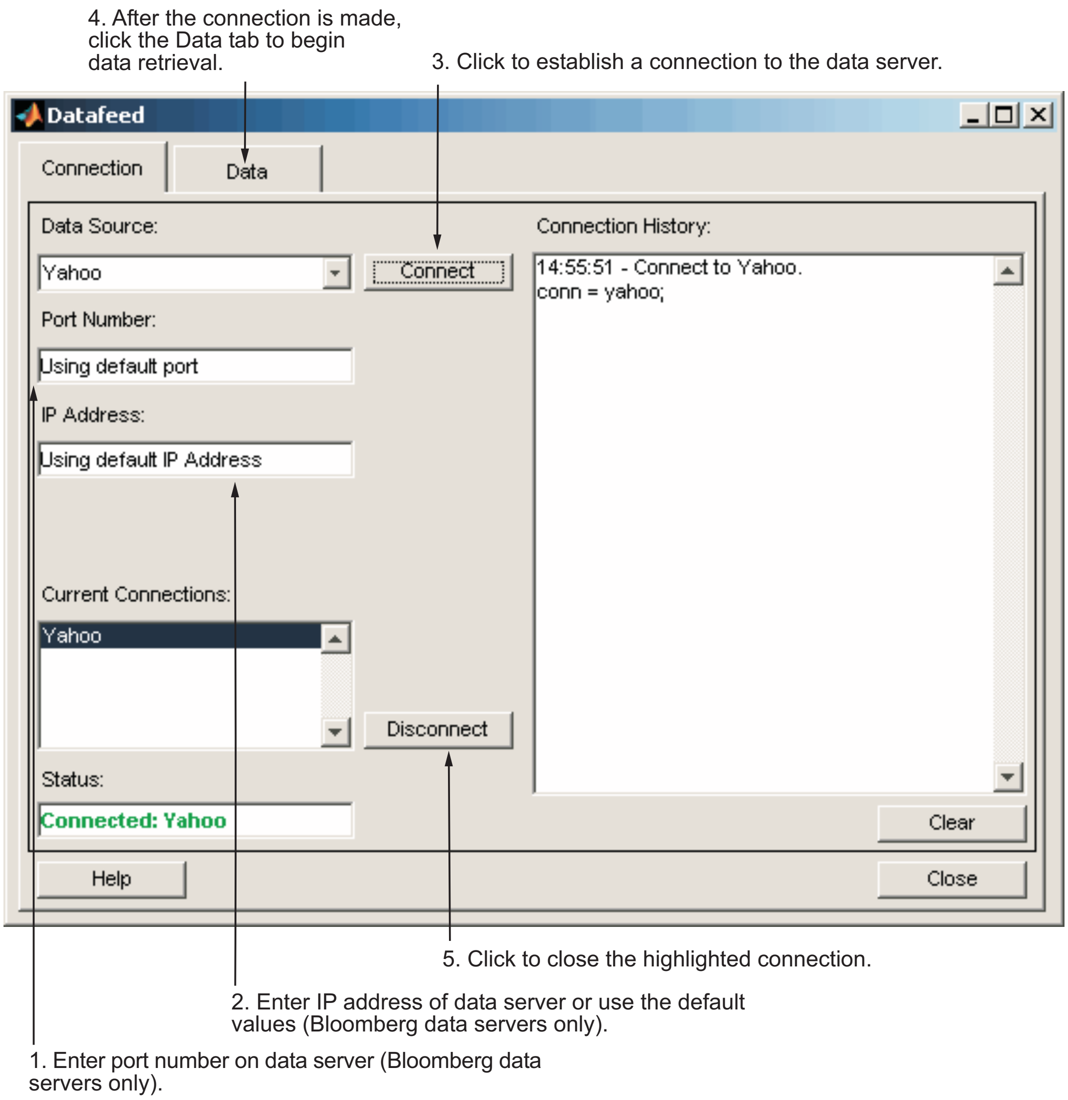
The Datafeed dialog box establishes the connection with the data server and manages data retrieval. Use this dialog box to connect to and retrieve data from Yahoo!

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

The Datafeed dialog box has 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 4-4.

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


4. After the connection is made, click the Data tab to begin data retrieval.
3. Click to establish a connection to the data server.
Datafeed
Connection
Data
Connect
Disconnect
Clear
Help
Close
5. Click to close the highlighted connection.
2. Enter IP address of data server or use the default values (Bloomberg data servers only).
1. Enter port number on data server (Bloomberg data servers only).

Data Source: Yahoo

Port Number: Using default port

IP Address: Using default IP Address

Current Connections: Yahoo

Status: Connected: Yahoo

Connection History:

```
14:55:51 - Connect to Yahoo.
conn = yahoo;
```

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 4-4.
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 in the previous procedure.

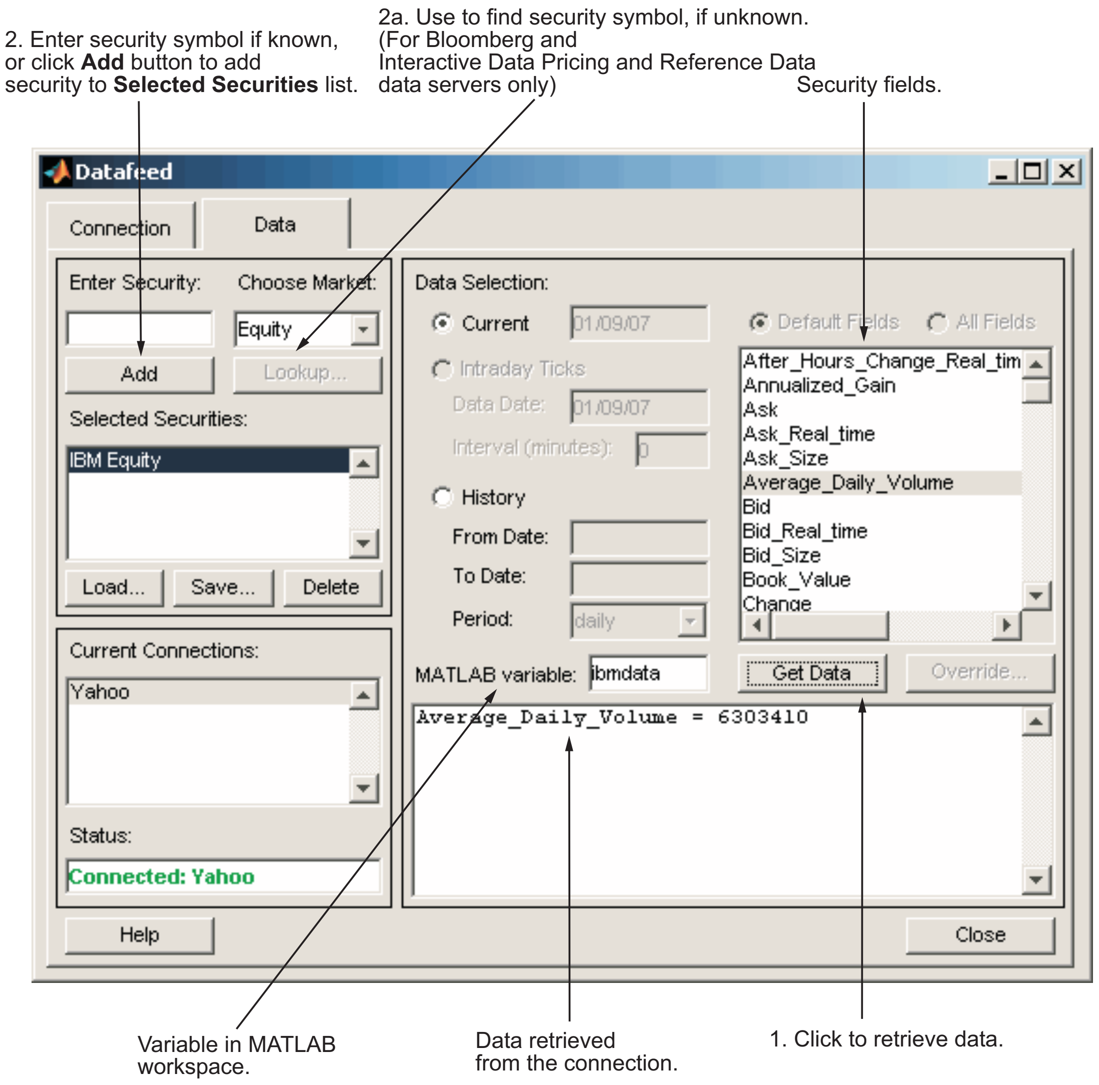
Tip You can also connect to the Bloomberg data server by clicking the **Connect** button and accepting the default values.

Retrieving Data

The **Data** tab lets you 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:
   - Click the **Default fields** button for the default set of data.
   - Click the **All fields** button for the full set of data.
4. Click the **Get Data** button to retrieve the data from the data server.

The following figure summarizes these steps.


2. Enter security symbol if known, or click **Add** button to add security to **Selected Securities** list.
2a. Use to find security symbol, if unknown. (For Bloomberg and Interactive Data Pricing and Reference Data data servers only)
Security fields.
Datafeed
Connection
Data
Add
Lookup...
Load...
Save...
Delete
Get Data
Override...
Help
Close
Variable in MATLAB workspace.
Data retrieved from the connection.
1. Click to retrieve data.

Enter Security:
Choose Market: Equity
Selected Securities: IBM Equity
Current Connections: Yahoo
Status: Connected: Yahoo

Data Selection:
[✓ Current] 01/09/07
Intraday Ticks
Data Date: 01/09/07
Interval (minutes): 0
History
From Date:
To Date:
Period: daily
[✓ Default Fields] All Fields

After_Hours_Change_Real_tim
Annualized_Gain
Ask
Ask_Real_time
Ask_Size
Average_Daily_Volume
Bid
Bid_Real_time
Bid_Size
Book_Value
Change

MATLAB variable: ibmdata

```
Average_Daily_Volume = 6303410
```

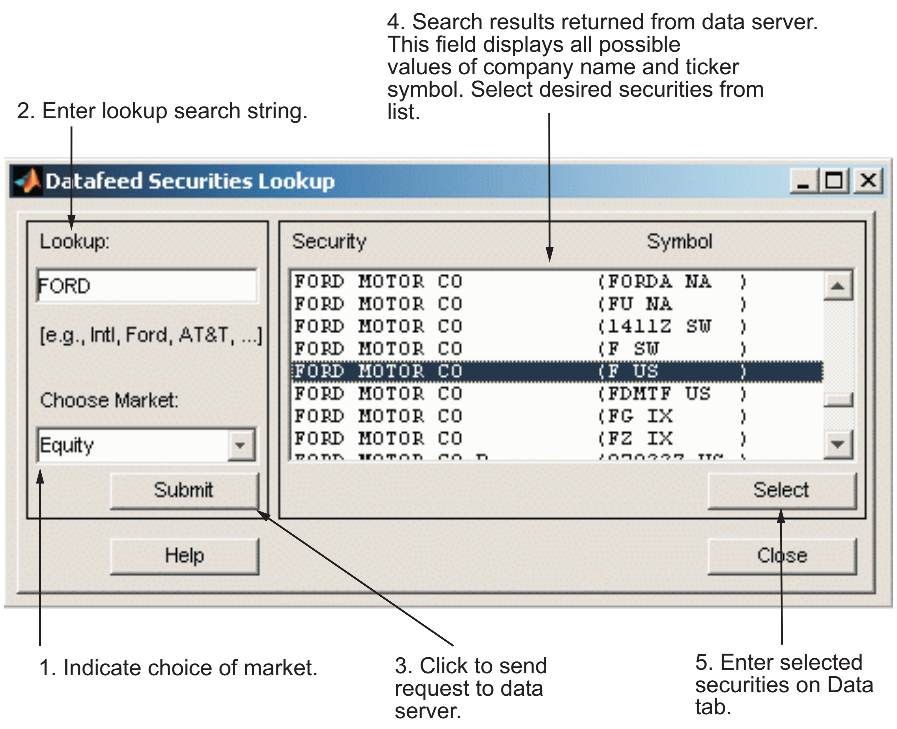
Obtain Ticker Symbol with Datafeed Securities Lookup

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
Select
Close
1. Indicate choice of market.
3. Click to send request to data server.
5. Enter selected securities on Data tab.

```
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      )
```

Functions — Alphabetical List

dftool

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 Bloomberg Desktop and Yahoo! connections.

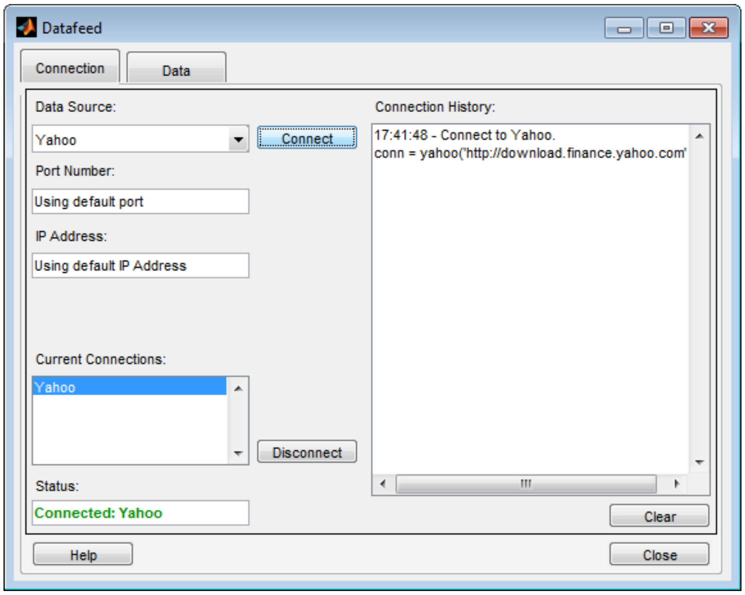
To display this dialog box, enter the `dftool` command in the Command Window.

The Datafeed dialog box has 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 4-4.

Examples

```
dftool
```



Data Source: Yahoo

Port Number: Using default port

IP Address: Using default IP Address

Current Connections: Yahoo

Status: Connected: Yahoo

Connection History:

```
17:41:48 - Connect to Yahoo.
conn = yahoo('http://download.finance.yahoo.com'
```

blp

Bloomberg Desktop connection V3

The `blp` function provides the connection to the Bloomberg Desktop.

There are other functions that connect to different Bloomberg services: Bloomberg Server (`blpsrv`), Bloomberg B-PIPE (`bpipe`), and Bloomberg Data License (`bdl`). For details about these Bloomberg services, see "Comparing Bloomberg Connections".

For the functions to run correctly, each function requires specific installation files. For details, see "Data Server Connection Requirements" on page 1-3.

Syntax

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

Description

`c = blp` connects to the Bloomberg Desktop. 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(portnumber,ip,timeout)` connects to Bloomberg Desktop using the IP address of the local machine where Bloomberg is running and a timeout value.

Examples

Connect to a Bloomberg Desktop

Establish a connection `c` to a Bloomberg Desktop.

```
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 Desktop with a Timeout

Establish a connection `c` 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.

- "Connect to Bloomberg"

Input Arguments

`portnumber` — 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 Desktop connection V3
connection object

Bloomberg Desktop connection V3, returned as a connection object with these properties.

Property	Description
`session`	Bloomberg V3 API COM object
`ipaddress`	IP address of the local machine
`port`	Port number of the local machine
`timeout`	Number in milliseconds specifying how long MATLAB attempts to connect to Bloomberg Desktop before timing out

More About

- "Data Server Connection Requirements" on page 1-3
- "Comparing Bloomberg Connections"
- "Workflow for Bloomberg"

See Also

`bdl` | `blpsrv` | `bpipe` | `category` | `close` | `fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

blpsrv

Bloomberg Server connection V3

The `blpsrv` function provides the connection to the Bloomberg Server.

There are other functions that connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg B-PIPE (`bpipe`), and Bloomberg Data License (`bdl`). For details about these Bloomberg services, see "Comparing Bloomberg Connections".

For the functions to run correctly, each function requires specific installation files. For details, see "Data Server Connection Requirements" on page 1-3.

Syntax

```
c = blpsrv(uuid,serverip)
c = blpsrv(uuid,serverip,portnumber)
c = blpsrv(uuid,serverip,portnumber,timeout)
```

Description

`c = blpsrv(uuid,serverip)` creates a Bloomberg Server connection `c` to the Bloomberg Server running on another machine. This machine is identified by IP address `serverip` using your Bloomberg UUID. You need a Bloomberg Server license for the machine running the Bloomberg Server.

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

`c = blpsrv(uuid,serverip,portnumber)` connects to the Bloomberg Server using a specific port number.

`c = blpsrv(uuid,serverip,portnumber,timeout)` connects to the Bloomberg Server using a timeout value.

Examples

Connect to the Bloomberg Server

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

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

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

c = blpsrv(uuid,serverip)
```

```
c = 

  blpsrv with properties:

         uuid: 12345678
         user: [1x1 com.bloomberglp.blpapi.impl.aT]
       userip: '111.11.11.112'
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: '111.11.11.111'
         port: 8194
      timeout: 0
```

`blpsrv` connects to the machine running the Bloomberg Server using the default port number `8194`. `blpsrv` creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID
- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg API object
- IP address of the machine running the Bloomberg Server
- Port number of the machine running the Bloomberg Server
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out

Close the Bloomberg Server connection.

```
close(c)
```

Connect to the Bloomberg Server with a Port Number

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

- The Bloomberg UUID is `12345678`.
- The IP address `serverip` for the machine running the Bloomberg Server is `'111.11.11.111'`.
- The default port number is `8194`.

```
uuid = 12345678;
serverip = '111.11.11.111';
portnumber = 8194;

c = blpsrv(uuid,serverip,portnumber)
```

```
c =

  blpsrv with properties:

         uuid: 12345678
         user: [1x1 com.bloomberglp.blpapi.impl.aT]
       userip: '111.11.11.112'
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: '111.11.11.111'
         port: 8194
      timeout: 0
```

`blpsrv` connects to the machine running the Bloomberg Server using the default port number `8194`. `blpsrv` creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID
- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg API object
- IP address of the machine running the Bloomberg Server

- Port number of the machine running the Bloomberg Server
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out

Close the Bloomberg Server connection.

```
close(c)
```

Connect to the Bloomberg Server with a Timeout

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

- The Bloomberg UUID is `12345678`.
- The IP address `serverip` for the machine running the Bloomberg Server is `'111.11.11.111'`.
- The port number is your default port number.
- The timeout value is 10 milliseconds.

```
uuid = 12345678;
serverip = '111.11.11.111';
portnumber = [];
timeout = 10;

c = blpsrv(uuid,serverip,portnumber,timeout)
```

```
c = 

  blpsrv with properties:

         uuid: 12345678
         user: [1x1 com.bloomberglp.blpapi.impl.aT]
       userip: '111.11.11.112'
      session: [1x1 com.bloomberglp.blpapi.Session]
    ipaddress: '111.11.11.111'
         port: 8194
      timeout: 10
```

`blpsrv` connects to the machine running the Bloomberg Server using the default port number `8194` and a timeout value of 10 milliseconds. `blpsrv` creates the Bloomberg Server connection object `c` with these properties:

- Bloomberg user identity UUID

- Bloomberg user identity object
- IP address of the machine running MATLAB
- Bloomberg API object
- IP address of the machine running the Bloomberg Server
- Port number of the machine running the Bloomberg Server
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out

Close the Bloomberg Server connection.

```
close(c)
```

- “Connect to Bloomberg”

Input Arguments

`uuid` — Bloomberg user identity UUID

scalar

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

Example: `12345678`

Data Types: `double`

`serverip` — Bloomberg Server IP address

string

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

Data Types: `char`

`portnumber` — Port number

`[]` (default) | scalar

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

Data Types: `double`

`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 Server connection V3

connection object

Bloomberg Server connection V3, returned as a Bloomberg Server connection object with these properties.

Property	Description
`uuid`	Bloomberg user identity UUID
`user`	Bloomberg user identity object
`userip`	IP address of the machine running MATLAB
`session`	Bloomberg API object
`ipaddress`	IP address of the machine running the Bloomberg Server
`port`	Port number of the machine running the Bloomberg Server
`timeout`	Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg Server before timing out

More About

- “Data Server Connection Requirements” on page 1-3
- “Comparing Bloomberg Connections”

- "Workflow for Bloomberg"

See Also

`bdl` | `blp` | `bpipe` | `category` | `close` | `fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

bpipe

Bloomberg B-PIPE connection V3

The `bpipe` function provides the connection to Bloomberg B-PIPE.

There are other functions that connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg Server (`blpsrv`), and Bloomberg Data License (`bdl`). For details about these Bloomberg services, see “Comparing Bloomberg Connections”.

For the functions to run correctly, each function requires specific installation files. For details, see “Data Server Connection Requirements” on page 1-3.

Syntax

```
c = bpipe(authtype,appname,serverip,portnumber)
c = bpipe(authtype,appname,serverip,portnumber,timeout)
```

Description

`c = bpipe(authtype,appname,serverip,portnumber)` creates a Bloomberg B-PIPE connection `c` using the following:

- Authorization type `authtype`
- Application name `appname`
- IP address `serverip` of the machine where the Bloomberg B-PIPE process is running
- Port number

`c = bpipe(authtype,appname,serverip,portnumber,timeout)` creates a Bloomberg B-PIPE connection `c` using a timeout value.

Examples

Create a Bloomberg B-PIPE Connection

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

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

```
authtype = 'OS_LOGON';
appname = '';
serverip = {'111.11.11.112'};
portnumber = 8194;

c = bpipe(authtype,appname,serverip,portnumber)
```

```
c =

  bpipe with properties:

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

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

- Application authentication type
- Bloomberg user authentication type
- Application name
- Bloomberg user identity object
- Bloomberg V3 API object
- IP address of the machine where the Bloomberg B-PIPE process is running
- Port number of the machine where the Bloomberg B-PIPE process is running
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg V3 B-PIPE API before timing out

Close the Bloomberg B-PIPE connection.

```
close(c)
```

Create a Bloomberg B-PIPE Connection with a Timeout

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

- The authentication is Windows Authentication when setting `authtype` to `'OS_LOGON'`.
- The application name is blank because you are not connecting to Bloomberg B-PIPE using an application.
- The IP address `serverip` for the machine, which is running the Bloomberg B-PIPE process, is `'111.11.11.112'`.
- The port number is `8194`.
- The timeout value is 1000 milliseconds.

```
authtype = 'OS_LOGON';
appname = '';
serverip = {'111.11.11.112'};
portnumber = 8194;
timeout = 1000;

c = bpipe(authtype,appname,serverip,portnumber,timeout)
```

```
c =

  bpipe with properties:

    appauthtype: ''
       authtype: 'OS_LOGON'
        appname: []
           user: [1x1 com.bloomberglp.blpapi.impl.aT]
        session: [1x1 com.bloomberglp.blpapi.Session]
      ipaddress: {'172.28.17.118'}
           port: 8194.00
        timeout: 1000.00
```

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

- Application authentication type

- Bloomberg user authentication type
- Application name
- Bloomberg user identity object
- Bloomberg V3 API object
- IP address of the machine where the Bloomberg B-PIPE process is running
- Port number of the machine where the Bloomberg B-PIPE process is running
- Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg V3 B-PIPE API before timing out

Close the Bloomberg B-PIPE connection.

```
close(c)
```

- "Connect to Bloomberg"

Input Arguments

`authtype` — Authorization type

string

Authorization type, specified as one of these enumerated Bloomberg strings.

Bloomberg String	Description
`'OS_LOGON'`	Create Bloomberg B-PIPE connection with Windows Authentication.
`'APPLICATION_ONLY'`	Create Bloomberg B-PIPE connection with application authentication.

For details, see the *Bloomberg B-PIPE API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `char`

`appname` — Application name

string

Application name, specified as a string to identify the application you are using that connects to Bloomberg B-PIPE.

Data Types: `char`

`serverip` — IP address for the machine
string | cell array

IP address for the machine, specified as a string or a cell array of strings. A string identifies the machine where the Bloomberg B-PIPE process is running, whereas a cell array of strings denotes multiple machines.

Data Types: `char` | `cell`

`portnumber` — Port number
`[ ]` (default) | scalar

Port number, specified as a scalar to identify the port number of the machine where the Bloomberg B-PIPE process is running.

Data Types: `double`

`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 B-PIPE connection
connection object

Bloomberg B-PIPE connection, returned as a connection object with these properties.

Property	Description
`appauthtype`	Application authentication type
`authtype`	Bloomberg user authentication type
`appname`	Application name
`user`	Bloomberg user identity object

Property	Description
`session`	Bloomberg V3 API object
`ipaddress`	IP address of the machine where the Bloomberg B-PIPE process is running
`port`	Port number of the machine where the Bloomberg B-PIPE process is running
`timeout`	Number in milliseconds specifying how long MATLAB attempts to connect to the machine running the Bloomberg V3 B-PIPE API before timing out

More About

- "Data Server Connection Requirements" on page 1-3
- "Comparing Bloomberg Connections"
- "Workflow for Bloomberg"

See Also

`bdl` | `blp` | `blpsrv` | `category` | `close` | `fieldinfo` | `fieldsearch` | `getdata` | `history` | `realtime` | `timeseries`

bdl

Bloomberg Data License connection

The `bdl` function provides the connection to the Bloomberg Data License.

There are other functions that connect to different Bloomberg services: Bloomberg Desktop (`blp`), Bloomberg Server (`blpsrv`), and Bloomberg B-PIPE (`bpipe`). For details about these Bloomberg services, see “Comparing Bloomberg Connections”.

For the functions to run correctly, each function requires specific installation files. For details, see “Data Server Connection Requirements” on page 1-3.

Syntax

```
c = bdl(username,password,hostname,portnumber,decrypt)
c = bdl(username,password,hostname,portnumber,decrypt,authtype,
keyfile,passphrase)
```

Description

`c = bdl(username,password,hostname,portnumber,decrypt)` connects to the Bloomberg Data License server using the Secure File Transfer Protocol (SFTP). `bdl` uses these input arguments:

- Bloomberg Data License SFTP server login name `username`
- Bloomberg Data License SFTP server password `password`
- Bloomberg Data License SFTP server name `hostname`
- Bloomberg Data License SFTP server port number `portnumber`
- Decryption code `decrypt`

`c = bdl(username,password,hostname,portnumber,decrypt,authtype, keyfile,passphrase)` connects to the Bloomberg Data License server using key authentication. Specify the full path to the key file `keyfile` and the pass phrase `passphrase`.

Examples

Connect to Bloomberg Data License Using Password Authentication

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

- The Bloomberg Data License SFTP server login name is `'dl111'`.
- The Bloomberg Data License SFTP server password is `'Pc=zXdA'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.

```
username = 'dl111';
password = 'Pc=zXdA';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'nAcLeZ';

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

```
c = 

bdl with properties:

          Login: 'dl111'
       Hostname: 'dlsftp.bloomberg.com'
           Port: 30206
     AuthOption: 'password'
        KeyFile: ''
     Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

`c` returns the Bloomberg Data License connection object with these properties:

- Bloomberg Data License SFTP server login name
- Bloomberg Data License SFTP server name
- Bloomberg Data License SFTP port number
- Authentication type is the default password authentication
- Key file is blank
- Bloomberg Data License API object

Close the Bloomberg Data License connection.

```
close(c)
```

Connect to Bloomberg Data License Using a Key File

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

- The Bloomberg Data License SFTP server login name is `'dl111'`.
- The Bloomberg Data License SFTP server password is `'Pc=zXdA'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.
- The authentication type is `'key'`.
- The full path to the key file is `'c:\temp\mykeyfile'`.
- The pass phrase is `'mykeyphrase'`.

```
username = 'dl111';
password = 'Pc=zXdA';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'nAcLeZ';
authtype = 'key';
keyfile = 'c:\temp\mykeyfile';
passphrase = 'mykeyphrase';

c = bdl(username,password,hostname,portnumber,decrypt,authtype,...
        keyfile,passphrase)

c =

bdl with properties:

          Login: 'dl111'
       Hostname: 'dlsftp.bloomberg.com'
           Port: 30206
     AuthOption: 'key'
        KeyFile: 'c:\temp\mykeyfile'
     Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

`c` returns the Bloomberg Data License connection object with these properties:

- Bloomberg Data License SFTP server login name
- Bloomberg Data License SFTP server name
- Bloomberg Data License SFTP port number
- Authentication type is key authentication
- Full path to the key file
- Bloomberg Data License API object

Close the Bloomberg Data License connection.

```
close(c)
```

- "Connect to Bloomberg"
- "Retrieve Data Using the Bloomberg Data License"

Input Arguments

`username` — User name
string

User name, specified as a string to denote your Bloomberg Data License SFTP server login name.

Data Types: `char`

`password` — Password
string

Password, specified as a string to denote your Bloomberg Data License SFTP server password.

Data Types: `char`

`hostname` — Server name
string

Server name, specified as a string to denote the Bloomberg Data License SFTP server name.

Data Types: `char`

`portnumber` — Port number
scalar

Port number, specified as a scalar to identify the Bloomberg Data License SFTP port number of the machine where the Bloomberg Data License server is running.

Data Types: `double`

`decrypt` — Decryption code
string

Decryption code, specified as a string to denote the DES encryption key.

Data Types: `char`

`authtype` — Authentication type
`'password'` (default) | `'key'`

Authentication type, specified as one of the preceding enumerated strings. If you specify `'password'`, you must supply the Bloomberg Data License SFTP server password. If you specify `'key'`, you must provide a key file name and a pass phrase.

Data Types: `char`

`keyfile` — Key file
string

Key file, specified as a string to denote the full path for the private key file. Use this argument only when authentication type authtype is `'key'`.

Data Types: `char`

`passphrase` — Pass phrase
string

Pass phrase, specified as a string. `bdl` uses this phrase to decrypt the key file. Use this argument only when authentication type authtype is `'key'`.

Data Types: `char`

Output Arguments

`c` — Bloomberg Data License connection
connection object

Bloomberg Data License connection, returned as a connection object with these properties.

Property	Description
`Login`	Bloomberg Data License SFTP server login name
`Hostname`	Bloomberg Data License SFTP server name
`Port`	Bloomberg Data License SFTP port number of the machine where the Bloomberg Data License server is running
`AuthOption`	Authentication type
`KeyFile`	Full path to the key file
`Connection`	Bloomberg Data License API object

More About

Tips

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

- "Data Server Connection Requirements" on page 1-3
- "Comparing Bloomberg Connections"
- "Workflow for Bloomberg"

See Also

`blp` | `blpsrv` | `bpipe` | `close` | `dir`

Introduced in R2015a

dir

Current Bloomberg Data License connection folder listing

Syntax

```
list = dir(c)
```

Description

`list = dir(c)` returns the contents of the current working folder using the Bloomberg Data License connection `c`.

Examples

Request the Bloomberg Data License Folder Listing

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

- The Bloomberg Data License SFTP server login name is `'dl111'`.
- The Bloomberg Data License SFTP server password is `'Pc=zXdA'`.
- The Bloomberg Data License SFTP server name is `'dlsftp.bloomberg.com'`.
- The Bloomberg Data License SFTP port number is `30206`.
- The decryption code is `'nAcLeZ'`.

```
username = 'dl111';
password = 'Pc=zXdA';
hostname = 'dlsftp.bloomberg.com';
portnumber = 30206;
decrypt = 'nAcLeZ';

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

Request the contents `list` of the current working folder using `c`.

```
list = dir(c)
list =

  'd--x--x--x    2 root     root         4096 Sep  5 11:25 bin'
  'dr--r--r--    2 root     root         4096 Sep  5 11:25 etc'
  'drwxrwxrwx    2 op       general     61440 Sep 24 02:11 notices'
  ...
```

`list` contains a cell array of strings. Each string is one folder or file name.

Close the Bloomberg Data License connection.

```
close(c)
```

- "Retrieve Data Using the Bloomberg Data License"

Input Arguments

`c` — Bloomberg Data License connection

connection object

Bloomberg Data License connection, specified as a connection object created using `bdl`.

Output Arguments

`list` — Current working folder contents

cell array

Current working folder contents, returned as an `n`-by-1 cell array of strings. `n` is the number of files or folders within the current working folder. Each string contains folder listing information for a folder or file name.

More About

Tips

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

- "Workflow for Bloomberg"

See Also

`bdl` | `close`

Introduced in R2015a

category

Field category search for Bloomberg connection V3

Syntax

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

Description

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

Examples

Search for the Bloomberg Last Price Field

Create the Bloomberg connection.

```
c = blp;
```

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

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'    'OP054'    'DELTA_LAST'    'Delta Last Trade...'    'Double'
    'Analysis'    'OP051'    'IVOL_LAST'     'Implied Volatili...'    'Double'
    'Analysis'    'OP006'    'DELTA'         'Delta Best Price'       'Double'
```

The columns in `d` contain the following:

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

Close the Bloomberg connection.

```
close(c)
```

Input Arguments

`c` — Bloomberg connection

connection object

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

`f` — Search term

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.

More About

- "Workflow for Bloomberg"

See Also

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

close

Close Bloomberg connection V3

Syntax

```
close(c)
```

Description

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

Examples

Close the Bloomberg Connection

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

```
c = blp;
```

Alternatively, you can establish these connections:

- Bloomberg Server API using `blpsrv`
- Bloomberg B-PIPE using `bpipe`
- Bloomberg Data License using `bdl`

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

```
close(c)
```

- “Connect to Bloomberg”
- “Retrieve Bloomberg Current Data”
- “Retrieve Bloomberg Historical Data”
- “Retrieve Bloomberg Intraday Tick Data”
- “Retrieve Bloomberg Real-Time Data”

Input Arguments

c — Bloomberg connection
connection object

Bloomberg connection, specified as one of these connection objects:

- Bloomberg connection V3 created using `blp`
- Bloomberg Server API connection created using `blpsrv`
- Bloomberg B-PIPE connection created using `bpipe`
- Bloomberg Data License connection created using `bdl`

More About

- "Workflow for Bloomberg"

See Also

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

eqs

Return equity screening data from Bloomberg connection V3

Syntax

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

Description

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

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

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

`d = eqs(c,sname,stype,languageid,group,Name,Value)` returns equity screening data additional options specified by the Name,Value pair argument.

Examples

Retrieve Equity Screening Data for a Screen

Create the Bloomberg connection.

```
c = blp;
```

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

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'        [1166249984.00]
    'South Africa'    'HARMONY GOLD MIN...'     'Mining'       [1239142656.00]

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

- Country name
- Company name
- Industry name
- Market capitalization
- Price
- Price-to-book ratio
- Price-earnings ratio
- Earnings per share

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data for a Screen Type

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve equity screening data for the screen called `Vehicle-Engine-Parts` and the screen type equal to `'GLOBAL'`.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL');
```

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

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

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'        'Price:D-1'    'P/E'
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36]    [17.28]

  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'      'EPS T12M'
    [           42.43]    [38248998912.00]    [    4.11]
    [           24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue

- Earnings per share

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data for a Screen in German

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve equity screening data for the screen called `Vehicle-Engine-Parts`, the screen type equal to `'GLOBAL'`, and return data in German.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','GERMAN');
```

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

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

```
  Columns 1 through 5

    'Ticker'        'Kurzname'          'Marktkapitalisie...'    'Preis:D-1'    'KGV'
    'HON     US'    'HONEYWELL INTL'    [     69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [     24799526912.00]    [   132.36]    [17.28]

  Columns 6 through 8

    'Gesamtertrag YTD'    'Erlös T12M'         'EPS T12M'
    [           42.43]    [38248998912.00]    [    4.11]
    [           24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers in German. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio
- Total return year-to-date

- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data for a Screen with a Specified Screen Folder Name

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve equity screening data for the Bloomberg screen called `Vehicle-Engine-Parts`, using the Bloomberg screen type `'GLOBAL'` and the language `'ENGLISH'`, and the Bloomberg screen folder name `'GENERAL'`.

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','ENGLISH','GENERAL');
```

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

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

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'        'Price:D-1'    'P/E'
    'HON     US'    'HONEYWELL INTL'    [69451382784.00]    [    88.51]    [16.81]
    'CMI     US'    'CUMMINS INC'       [24799526912.00]    [   132.36]    [17.28]

  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'      'EPS T12M'
    [           42.43]    [38248998912.00]    [    4.11]
    [           24.43]    [17004999936.00]    [    7.57]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization

- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data Using Override Fields

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve equity screening data as of a specified date using these input arguments. The override field `PiTDate` is equivalent to the flag `AsOf` in the Bloomberg Excel Add-In.

- Bloomberg connection `c`
- Bloomberg screen is `Vehicle-Engine-Parts`
- Bloomberg screen type is `'GLOBAL'`
- Language is `'ENGLISH'`
- Bloomberg screen folder name is `'GENERAL'`
- Override field `PiTDate` is September 9, 2014

```
d = eqs(c,'Vehicle-Engine-Parts','GLOBAL','ENGLISH','GENERAL',...
        'overrideFields',{'PiTDate','20140909'});
```

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

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

```
ans =

  Columns 1 through 5

    'Ticker'        'Short Name'        'Market Cap'    'Price:D-1'       'P/E'
    'HON US'     'HONEYWELL INTL'      [7.3919e+10]    [   94.4600]    [17.8087]
    'TSLA US'    'TESLA MOTORS'       [3.4707e+10]    [ 278.4800]    [    NaN]

  Columns 6 through 8

    'Total Return YTD'    'Revenue T12M'    'EPS T12M'
    [          4.8907]    [  3.9966e+10]    [  5.1600]
    [         85.1239]    [  2.4365e+09]    [ -1.3500]
```

`d` contains Bloomberg equity screening data for the `Vehicle-Engine-Parts` screen as of September 9, 2014. The first row contains column headers. The subsequent rows contain the returned data. The columns in `d` are:

- Ticker symbol
- Company name
- Market capitalization
- Price
- Price-earnings ratio
- Total return year-to-date
- Revenue
- Earnings per share

Close the connection.

```
close(c)
```

Input Arguments

`c` — Bloomberg connection

connection object

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

`sname` — Screen name

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`

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: `'overrideFields',{'PiTDate','20140909'}`

`'overrideFields'` — Override fields
cell array

Override fields, specified as the comma-separated pair consisting of `'overrideFields'` and an `n`-by-`2` cell array. The first column of the cell array is the override field. The second column is the override value.

Data Types: `cell`

Output Arguments

`d` — Return data
cell array

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

More About

- “Workflow for Bloomberg”

See Also

`blp` | `close` | `getdata` | `tahistory`

fieldinfo

Field information for Bloomberg connection V3

Syntax

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

Description

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

Examples

Retrieve Information for the Bloomberg Last Price Field

Create a Bloomberg connection.

```
c = blp;
```

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

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 trade Monday through Friday, this field will be pop
...
```

```
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)
```

Input Arguments

`c` — Bloomberg connection

connection object

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

`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.

More About

- "Workflow for Bloomberg"

See Also

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

fieldsearch

Field search for Bloomberg connection V3

Syntax

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

Description

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

Examples

Search for the Bloomberg Last Price Field

Create a Bloomberg connection.

```
c = blp;
```

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

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 Price'            'Double'
    'Market Activity/...'    'RQ005'    'LAST_PRICE'           'Last Trade/Last ...'   'Double'
    'Market Activity/...'    'RQ134'    'LAST_ALL_SESSIONS'    'Last Price All S...'   'Double'
```

The columns in `d` contain the following:

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

Close the Bloomberg connection.

```
close(c)
```

Input Arguments

`c` — Bloomberg connection
connection object

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

`f` — Search term
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.

More About

- "Workflow for Bloomberg"

See Also

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

get

Properties of Bloomberg connection V3

Syntax

```
v = get(c)
v = get(c,properties)
```

Description

`v = get(c)` returns a structure where each field name is the name of a property of `c` and each field contains the value of that property.

`v = get(c,properties)` returns the value of the specified properties `properties` for the Bloomberg V3 connection object.

Examples

Retrieve Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the Bloomberg connection properties.

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

```
v = 

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

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve One Bloomberg Connection Property

Create the Bloomberg connection.

```
c = blp;
```

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

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;
```

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

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)
```

Input Arguments

c — Bloomberg connection

connection object

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

`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.

More About

- "Workflow for Bloomberg"

See Also

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

getbulkdata

Bulk data with header information for Bloomberg connection V3

Syntax

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

Description

`d = getbulkdata(c,s,f)` returns the bulk data for the fields `f` for the security list `s`.

`d = getbulkdata(c,s,f,o,ov)` returns the bulk data using the override fields `o` with corresponding override values `ov`.

`d = getbulkdata(c,s,f,o,ov,Name,Value)` returns the bulk data with additional options specified by one or more `Name`,`Value` pair arguments for additional Bloomberg request settings.

`[d,sec] = getbulkdata( ___ )` additionally returns the security list `sec` using any of the input arguments in the previous syntaxes.

Examples

Return a Specific Field for a Given Security

Create the Bloomberg connection.

```
c = blp;
```

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

Return the dividend history for IBM.

```
security = 'IBM US Equity';
```

```
field = 'DVD_HIST'; % Dividend history field

[d,sec] = getbulkdata(c,security,field)

d =

    DVD_HIST: {{149x7 cell}}

sec =

    'IBM US Equity'
```

`d` is a structure with one field that contains a cell array with the returned bulk data. `sec` contains the IBM security name.

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}

ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       735536]    [ 735544]    [     735546]    [      735578]    [           0.95]    'Quarter'
    [       735445]    [ 735453]    [     735455]    [      735487]    [           0.95]    'Quarter'
    [       735354]    [ 735362]    [     735364]    [      735395]    [           0.95]    'Quarter'
    ...

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    ...
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

Return a Specific Field Using Override Values

Create the Bloomberg connection.

```
c = blp;
```

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

Return the dividend history for IBM with dividend dates from January 1, 2004 through January 1, 2005.

```
security = 'IBM US Equity';
field = 'DVD_HIST';                            % Dividend history field
override = {'DVD_START_DT','DVD_END_DT'};  % Dividend start and
                                           % End dates
overridevalues = {'20040101','20050101'};

[d,sec] = getbulkdata(c,security,field,override,overridevalues)
```

```
d =

    DVD_HIST: {{5x7 cell}}

sec =

    'IBM US Equity'
```

`d` is a structure with one field that contains a cell array with the returned bulk data. `sec` contains the IBM security name.

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}
```

```
ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       732246]    [ 732259]    [     732261]    [      732291]    [           0.18]    'Quarter'
    [       732155]    [ 732165]    [     732169]    [      732200]    [           0.18]    'Quarter'
    [       732064]    [ 732073]    [     732077]    [      732108]    [           0.18]    'Quarter'
    [       731973]    [ 731983]    [     731987]    [      732016]    [           0.16]    'Quarter'

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

Return a Specific Field Using Name-Value Pair Arguments

Create the Bloomberg connection.

```
c = blp;
```

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

Return the closing price and dividend history for IBM with dividend dates from January 1, 2004 through January 1, 2005. Specify the data return format as a string by setting the name-value pair argument `'returnFormattedValue'` to `'true'`.

```
security = 'IBM US Equity';
fields = {'LAST_PRICE','DVD_HIST'};          % Closing price and
                                             % Dividend history fields
override = {'DVD_START_DT','DVD_END_DT'};    % Dividend start and
                                             % End dates
overridevalues = {'20040101','20050101'};

[d,sec] = getbulkdata(c,security,fields,override,overridevalues,...
                      'returnFormattedValue',true)

d = 

      DVD_HIST: {{5x7 cell}}
    LAST_PRICE: {'188.74'}

sec = 

    'IBM US Equity'
```

`d` is a structure with two fields. The first field `DVD_HIST` contains a cell array with the dividend historical data as a cell array. The second field `LAST_PRICE` contains a cell array with the closing price as a string. `sec` contains the IBM security name.

Display the closing price.

```
d.LAST_PRICE
```

```
ans =

    '188.74'
```

Display the dividend history with the associated header information by accessing the structure field `DVD_HIST`. This field is a cell array that contains one cell array. The nested cell array contains the dividend history data. Access the contents of the nested cell using cell array indexing.

```
d.DVD_HIST{1}
```

```
ans =

  Columns 1 through 6

    'Declared Date'    'Ex-Date'    'Record Date'    'Payable Date'    'Dividend Amount'    'Dividend Frequency'
    [       732246]    [ 732259]    [     732261]    [      732291]    [           0.18]    'Quarter'
    [       732155]    [ 732165]    [     732169]    [      732200]    [           0.18]    'Quarter'
    [       732064]    [ 732073]    [     732077]    [      732108]    [           0.18]    'Quarter'
    [       731973]    [ 731983]    [     731987]    [      732016]    [           0.16]    'Quarter'

  Column 7

    'Dividend Type'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
    'Regular Cash'
```

The first row of the dividend history data is the header information that describes the contents of each column.

Close the connection.

```
close(c)
```

Input Arguments

c — Bloomberg connection

connection object

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

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 string specific to Bloomberg for one data field or a cell array of strings specific to Bloomberg 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 string specific to Bloomberg for one data field or a cell array of strings specific to Bloomberg 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.

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: `'returnFormattedValue',true`

`'returnEids'` — Entitlement identifiers
true | false

Entitlement identifiers, specified as a Boolean. `true` adds a name and value for the entitlement identifier (EID) date to the return data.

Data Types: `logical`

`'returnFormattedValue'` — Return format
true | false

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

Data Types: `logical`

`'useUTCTime'` — Date time format
true | false

Date time format, specified as a Boolean. `true` returns date and time values as Coordinated Universal Time (UTC) and `false` defaults to the Bloomberg **TZDF <GO>** settings of the requestor.

Data Types: `logical`

`'forcedDelay'` — Latest reference data
true | false

Latest reference data, specified as a Boolean. `true` returns the latest data up to the delay period specified by the exchange for the security.

Data Types: `logical`

Output Arguments

d — Bloomberg 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`

More About

- "Workflow for Bloomberg"

See Also

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

getdata

Current data for Bloomberg connection V3

Syntax

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

Description

`d = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`. `getdata` accesses the Bloomberg reference data service.

`d = getdata(c,s,f,o,ov)` returns the data using the override fields `o` with corresponding override values `ov`.

`d = getdata(c,s,f,o,ov,Name,Value)` returns the data using `Name`,`Value` pair arguments for additional Bloomberg request settings.

`[d,sec] = getdata(___)` additionally returns the security list `sec` using any of the input arguments in the previous syntaxes.

Examples

Return the Last and Open Price of the Given Security

Create the Bloomberg connection.

```
c = blp;
```

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

Request last and open prices for Microsoft.

```
[d,sec] = getdata(c,'MSFT US Equity',{'LAST_PRICE';'OPEN'})

d = 
    LAST_PRICE: 33.3401
          OPEN: 33.6000

sec = 
    'MSFT US Equity'
```

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

Close the connection.

```
close(c)
```

Return the Requested Fields Given Override Fields and Values

Create the Bloomberg connection.

```
c = blp;
```

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

Request data for Bloomberg fields `'YLD_YTM_ASK'`, `'ASK'`, and `'OAS_SPREAD_ASK'` when the Bloomberg field `'OAS_VOL_ASK'` is `'14.000000'`.

```
[d,sec] = getdata(c,'030096AF8 Corp',...
   {'YLD_YTM_ASK','ASK','OAS_SPREAD_ASK','OAS_VOL_ASK'},...
   {'OAS_VOL_ASK'},{'14.000000'})

d = 
       YLD_YTM_ASK: 5.6763
               ASK: 120.7500
    OAS_SPREAD_ASK: 307.9824
       OAS_VOL_ASK: 14

sec = 
    '030096AF8 Corp'
```

`getdata` returns a structure `d` with the resulting values for the requested fields.

Close the connection.

```
close(c)
```

Return a Request for a Security Using its CUSIP Number

Create the Bloomberg connection.

```
c = blp;
```

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

Request the last price for IBM with the CUSIP number.

```
d = getdata(c,'/cusip/459200101','LAST_PRICE')
```

```
d = 
    LAST_PRICE: 182.5100
```

`getdata` returns a structure `d` with the last price.

Close the connection.

```
close(c)
```

Return the Last Price for the Security with a Pricing Source

Create the Bloomberg connection.

```
c = blp;
```

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

Specify IBM with the CUSIP number and the pricing source `BGN` after the `@` symbol.

```
d = getdata(c,'/cusip/459200101@BGN','LAST_PRICE')
```

```
d = 
    LAST_PRICE: 186.81
```

`getdata` returns a structure `d` with the last price.

Close the connection.

```
close(c)
```

Return the Constituent Weights Using a Date Override

Create the Bloomberg connection.

```
c = blp;
```

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

Return the constituent weights for the Dow Jones Index as of January 1, 2010 using a date override with the required date format `YYYYMMDD`.

```
d = getdata(c,'DJX Index','INDX_MWEIGHT','END_DT','20100101')
```

```
d =
    INDX_MWEIGHT: {{30x2 cell}}
```

`getdata` returns a structure `d` with a cell array where the first column is the index and the second column is the constituent weight.

Display the constituent weights for each index.

```
d.INDX_MWEIGHT{1,1}
```

```
ans =
    'AA UN'      [1.1683]
    'AXP UN'     [2.9366]
    'BA UN'      [3.9229]
    'BAC UN'     [1.0914]
    ...
```

Close the connection.

```
close(c)
```

- “Retrieve Bloomberg Current Data”

Input Arguments

c — Bloomberg connection

connection object

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

`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 string specific to Bloomberg for one data field or a cell array of strings specific to Bloomberg 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 string specific to Bloomberg for one data field or a cell array of strings specific to Bloomberg 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.

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. `true` adds a name and value for the entitlement identifier (EID) date to the return data.

Data Types: `logical`

`'returnFormattedValue'` — Return format

true | false

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

Data Types: `logical`

`'useUTCTime'` — Date time format

true | false

Date time format, specified as a Boolean. `true` returns date and time values as Coordinated Universal Time (UTC) and `false` defaults to the Bloomberg **TZDF <GO>** settings of the requestor.

Data Types: `logical`

`'forcedDelay'` — Latest reference data

true | false

Latest reference data, specified as a Boolean. `true` returns the latest data up to the delay period specified by the exchange for the security.

Data Types: `logical`

Output Arguments

`d` — Bloomberg 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`

More About

Tips

- Bloomberg V3 data supports additional name-value pair arguments. To access further information on these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

- You can check data and field availability by using the Bloomberg Excel® Add-In.
- “Workflow for Bloomberg”

See Also

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

history

Historical data for Bloomberg connection V3

Syntax

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

Description

`d = history(c,s,f,fromdate,todate)` returns the historical data for the security list `s` and the connection object `c` for the fields `f` for the dates `FromDate` through `ToDate`. Date strings can be input in any format recognized by MATLAB. `sec` is the security list that maps the order of the return data. The return data `d` is sorted to match the input order of `s`.

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

`d = history(c,s,f,fromdate,todate,period,currency)` returns the historical data for the security list `s` for the fields `f` and the dates `fromdate` through `todate` based on the given currency `currency`.

`d = history(c,s,f,fromdate,todate,period,currency,Name,Value)` returns the historical data for the security list `s` using additional options specified by one or more Name,Value pair arguments.

`[d,sec] = history( ___ )` additionally returns the security list `sec` using any of the input arguments in the previous syntaxes. The return data, `d` and `sec`, are sorted to match the input order of `s`.

Examples

Retrieve the Daily Closing Price for a Date Range

Create the Bloomberg connection.

```
c = blp;
```

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

Get the daily closing price from August 1, 2010 through August 10, 2010 for the IBM security.

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

```
d =

     734352.00        123.55
     734353.00        123.18
     734354.00        124.03
     734355.00        124.56
     734356.00        123.58
     734359.00        125.34
     734360.00        125.19


sec = 

    'IBM US Equity'
```

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve the Monthly Closing Price for a Date Range

Create the Bloomberg connection.

```
c = blp;
```

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

Get the monthly closing price from August 1, 2010 through December 10, 2010 for the IBM security.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','12/10/2010','monthly')
```

```
d =

     734360.00        125.19
     734391.00        121.53
     734421.00        131.85
     734452.00        139.78
     734482.00        138.13


sec =

    'IBM US Equity'
```

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

Close the Bloomberg connection.

```
close(c)
```

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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;
```

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

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;
```

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

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;
```

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

Get the daily closing price from August 1, 2010 through August 10, 2010 for the IBM security in U.S. currency `'USD'`. The prices are adjusted for normal cash and splits.

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

```
d =

     734352.00        123.55
     734353.00        123.18
     734354.00        124.03
     734355.00        124.56
     734356.00        123.58
     734359.00        125.34
     734360.00        125.19


sec =

    'IBM US Equity'
```

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

Close the Bloomberg connection.

```
close(c)
```

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

Create the Bloomberg connection.

```
c = blp;
```

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

Get the daily closing price from January 1, 2012 through January 1, 2013 for the security specified with a CUSIP number `/cusip/459200101` and with pricing source `BGN`.

```
d = 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;
```

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

Return the closing price for the given dates in international format for the security `'MSFT@BGN US Equity'`.

```
stDt = datenum('01/06/11','dd/mm/yyyy');
endDt = datenum('01/06/12','dd/mm/yyyy');
[d,sec] = history(c,'MSFT@BGN US Equity','LAST_PRICE',...
                  stDt,endDt,{'previous_value','all_calendar_days'})
```

```
d =

     734655.00         22.92
     734656.00          22.72
     734657.00          22.42
     ...

sec =

    'MSFT@BGN US Equity'
```

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve the Median Estimated Earnings Per Share Using Override Fields

Create the Bloomberg connection.

```
c = blp;
```

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

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

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)
```

- "Retrieve Bloomberg Historical Data"

Input Arguments

`c` — Bloomberg connection
connection object

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

`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'}`, this function returns daily data for all calendar days reporting missing data as `NaN`s. When `period` is set to `{'actual'}`, this 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. This table shows the values for `period`.

Value	Description
`'daily'`	Return data for each day.
`'weekly'`	Return data for each week.
`'monthly'`	Return data for each month.
`'quarterly'`	Return data for each quarter.
`'semi_annually'`	Return data semiannually.
`'yearly'`	Return data for each year.
`'actual'`	Anchor date specification for an actual date. The anchor date is the date to which all other reported dates are related. For this function, for periodicities other than daily, `enddate` is the anchor date. For example, if you set the period to weekly and the `enddate` 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 `enddate` is the 20th of a month, each reported data point would be for the 20th of each month in the date range.
`'calendar'`	Anchor date specification for a calendar year.
`'fiscal'`	Anchor date specification for a fiscal year.
`'non_trading_weekdays'`	Return data for all weekdays.
`'all_calendar_days'`	Return data for all calendar days.
`'active_days_only'`	Return data for only active trading days.
`'previous_value'`	Fill missing values with previous values for dates without trading activity for the security.

Value	Description
`'nil_value'`	Fill missing values with a `NaN` for dates without trading activity for the security.

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`

`'overrideFields'` — Override fields

cell array

Override fields, specified as the comma-separated pair consisting of `'overrideFields'` and an `n`-by-`2` cell array. The first column of the cell array is the override field and the second column is the override value.

Example: `'overrideFields',`
`{'IVOL_DELTA_LEVEL','DELTA_LVL_10';'IVOL_DELTA_PUT_OR_CALL','IVOL_PUT';'IVOL_M`

Data Types: `cell`

`'adjustmentNormal'` — Historical normal pricing adjustment

true | false

Historical normal pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentNormal'` and a Boolean to reflect:

- Regular Cash
- Interim
- 1st Interim
- 2nd Interim
- 3rd Interim
- 4th Interim
- 5th Interim
- Income
- Estimated
- Partnership Distribution
- Final
- Interest on Capital
- Distribution
- Prorated

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

`'adjustmentAbnormal'` — Historical abnormal pricing adjustment

true | false

Historical abnormal pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentAbnormal'` and a Boolean to reflect:

- Special Cash
- Liquidation
- Capital Gains
- Long-Term Capital Gains
- Short-Term Capital Gains
- Memorial
- Return of Capital
- Rights Redemption
- Miscellaneous
- Return Premium
- Preferred Rights Redemption
- Proceeds/Rights
- Proceeds/Shares
- Proceeds/Warrants

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

`'adjustmentSplit'` — Historical split pricing or volume adjustment

true | false

Historical split pricing or volume adjustment, specified as the comma-separated pair consisting of `'adjustmentSplit'` and a Boolean to reflect:

- Spin-Offs
- Stock Splits/Consolidations
- Stock Dividend/Bonus
- Rights Offerings/Entitlement

For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

`'adjustmentFollowDPDF'` — Historical pricing adjustment
true (default) | false

Historical pricing adjustment, specified as the comma-separated pair consisting of `'adjustmentFollowDPDF'` and a Boolean. Setting this name-value pair follows the **DPDF <GO>** option from the Bloomberg terminal. For details about these additional name-value pairs, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: `logical`

Output Arguments

`d` — Bloomberg 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`

More About

Tips

- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/javaclasspath.txt`. For details about the static Java class path, see “Static Path”.
- You can check data and field availability by using the Bloomberg Excel Add-In.

- “Workflow for Bloomberg”

See Also

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

isconnection

Determine Bloomberg connection V3

Syntax

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

Description

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

Examples

Validate the Bloomberg Connection

Create the Bloomberg connection.

```
c = blp;
```

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

Validate the Bloomberg connection.

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

```
v =

     1
```

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

Close the Bloomberg connection.

```
close(c)
```

- “Connect to Bloomberg”

Input Arguments

c — Bloomberg connection
connection object

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

Output Arguments

v — Valid Bloomberg connection
true | false

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

More About

- "Workflow for Bloomberg"

See Also

`blp` | `blpsrv` | `bpipe` | `close` | `getdata`

lookup

Lookup to find information about securities for Bloomberg connection V3

Syntax

```
l = lookup(c,q,reqtype,Name,Value)
```

Description

`l = lookup(c,q,reqtype,Name,Value)` retrieves data based on criteria in the query q for a specific request type reqtype using the Bloomberg connection c. For additional information about the query criteria and the possible name-value pair combinations, see the *Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Examples

Look Up 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;
```

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

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;
```

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

Retrieve the curve data for the credit default swap subtype of corporate bonds for a `'GOLD'` related curve `'CD1016'`. Return a maximum of `10` rows of data for the U.S. with `'USD'` currency.

```
curves = lookup(c,'GOLD','curveListRequest','maxResults',10,...
                'countryCode','US','currencyCode','USD',...
                'curveid','CD1016','type','CORP','subtype','CDS')
```

```
curves = 

            curve: {'YCCD1016 Index'}
      description: {'Goldman Sachs Group Inc/The'}
          country: {'US'}
         currency: {'USD'}
          curveid: {'CD1016'}
             type: {'CORP'}
          subtype: {'CDS'}
```

```
        publisher: {'Bloomberg'}
            bbgid: {''}
```

One row of data displays as Bloomberg curve name `'YCCD1016 Index'` with Bloomberg description `'Goldman Sachs Group Inc/The'` in the U.S. with `'USD'` currency. The Bloomberg short-form identifier for the curve is `'CD1016'`. Bloomberg is the publisher and the `bbgid` is blank.

Close the Bloomberg connection.

```
close(c)
```

Look Up 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;
```

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

Filter government security data with ticker filter of `'T'` for a maximum of `10` rows of data.

```
govts = lookup(c,'T','govtListRequest','maxResults',10,...
               'partialMatch',false)
```

```
govts = 

      parseky: {10x1 cell}
         name: {10x1 cell}
       ticker: {10x1 cell}
```

The Government Security Lookup returns `parseky` data, the name and ticker of the United States Treasury bonds.

Display the `parseky` data.

```
govts.parseky
```

```
ans = 
    '912828VS Govt'
    '912828RE Govt'
    '912810RC Govt'
    '912810RB Govt'
    '912828VU Govt'
    '912828VV Govt'
    '912828VB Govt'
    '912828VR Govt'
    '912828VW Govt'
    '912828VQ Govt'
```

Display the names of the United States Treasury bonds.

```
govts.name
```

```
ans = 
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
    'United States Treasury Note/Bond'
```

Display the tickers of the United States Treasury bonds.

```
govts.ticker
```

```
ans = 
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
    'T'
```

Close the Bloomberg connection.

```
close(c)
```

Input Arguments

c — Bloomberg connection

connection object

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

q — Keyword query

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
`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

More About

- "Workflow for Bloomberg"

See Also

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

realtime

Real-time data for Bloomberg connection V3

Syntax

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

Description

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

`[subs,t] = realtime(c,s,f,eventhandler)` returns the subscription list `subs` and the timer `t` associated with the real-time event handler for the subscription list. Given connection `c`, the `realtime` function subscribes to a security or securities `s` and requests fields `f`, to update in real time while running an event handler `eventhandler`.

Examples

Retrieve Data for One Security

Retrieve a snapshot of data for one security only.

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the last trade and volume of the IBM security.

```
d = realtime(c,'IBM US Equity',{'Last_Trade','Volume'})
```

```
d = 

    LAST_TRADE: '181.76'
        VOLUME: '7277793'
```

Close the Bloomberg connection.

```
close(c)
```

Retrieve Data for One Security Using the Event Handler `v3stockticker`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3stockticker` that returns Bloomberg stock tick data.

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the last trade and volume for the IBM security using the event handler `v3stockticker`.

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

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

```
subs = 

com.bloomberglp.blpapi.SubscriptionList@79f07684

   Timer Object: timer-2

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

   Callbacks
```

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

** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

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

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve Data for Multiple Securities Using the Event Handler `v3stockticker`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3stockticker` that returns Bloomberg stock tick data.

Create the Bloomberg connection.

```
c = blp;
```

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

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

`v3stockticker` requires the input argument `f` of `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 last trade price and volume.

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve Data for One Security Using the Event Handler `v3showtrades`

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3showtrades` that creates a figure showing requested data for a security.

Create the Bloomberg connection.

```
c = blp;
```

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

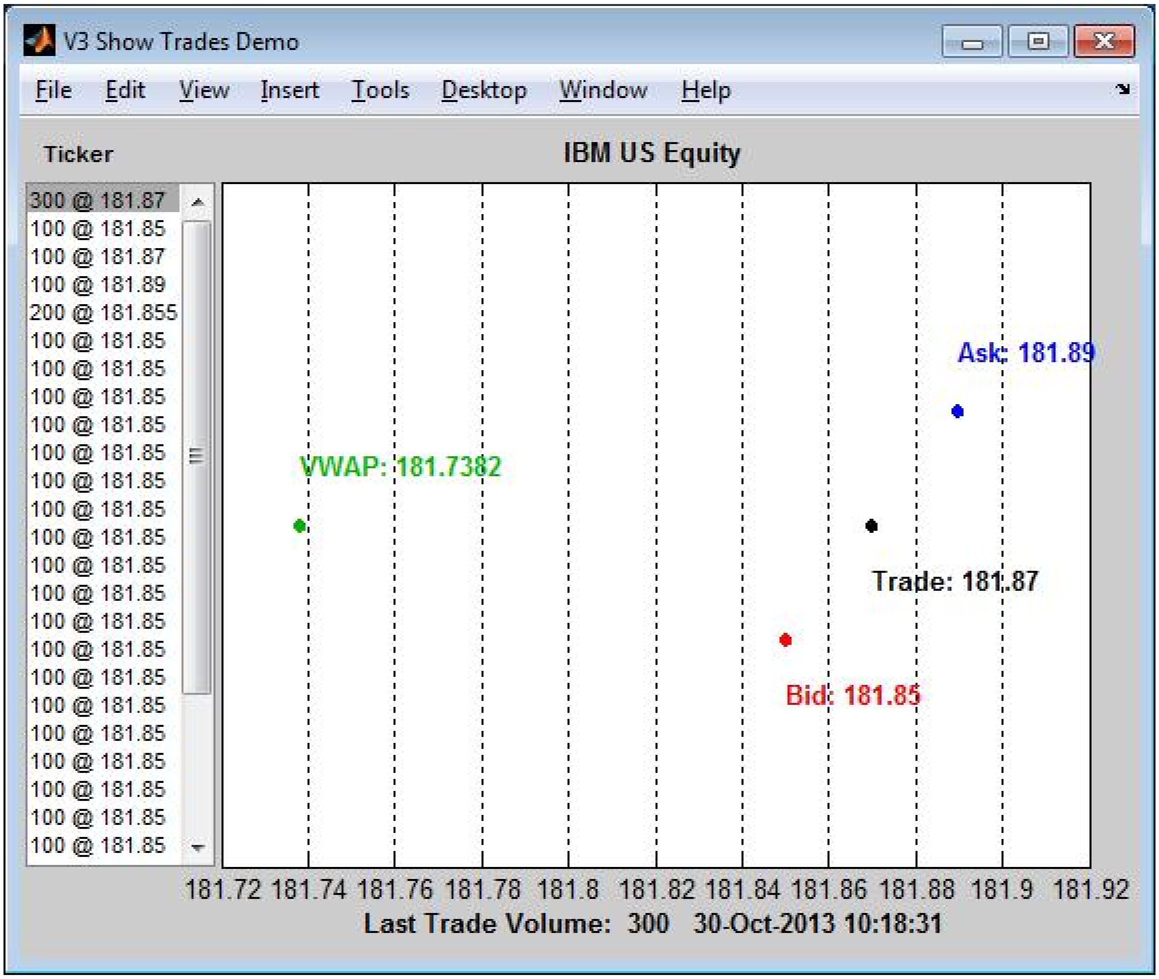
Retrieve volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for the IBM security using the event handler `v3showtrades`.

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

```
[subs,t] = realtime(c,'IBM US Equity',...
                    {'Last_Trade','Bid','Ask','Volume','VWAP'},...
                    'v3showtrades')
```

```
subs =

com.bloomberglp.blpapi.SubscriptionList@5c17dcdb


   Timer Object: timer-4

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

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

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `v3showtrades` displays a figure showing volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for IBM.



```
300 @ 181.87
100 @ 181.85
100 @ 181.87
100 @ 181.89
200 @ 181.855
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
100 @ 181.85
```

Last Trade Volume: 300 30-Oct-2013 10:18:31

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve Data for One Security Using the Event Handler v3pricevol

You can create your own event handler function to process Bloomberg data. For this example, use the event handler `v3pricevol` that creates a figure showing last price and volume data for a security.

Create the Bloomberg connection.

```
c = blp;
```

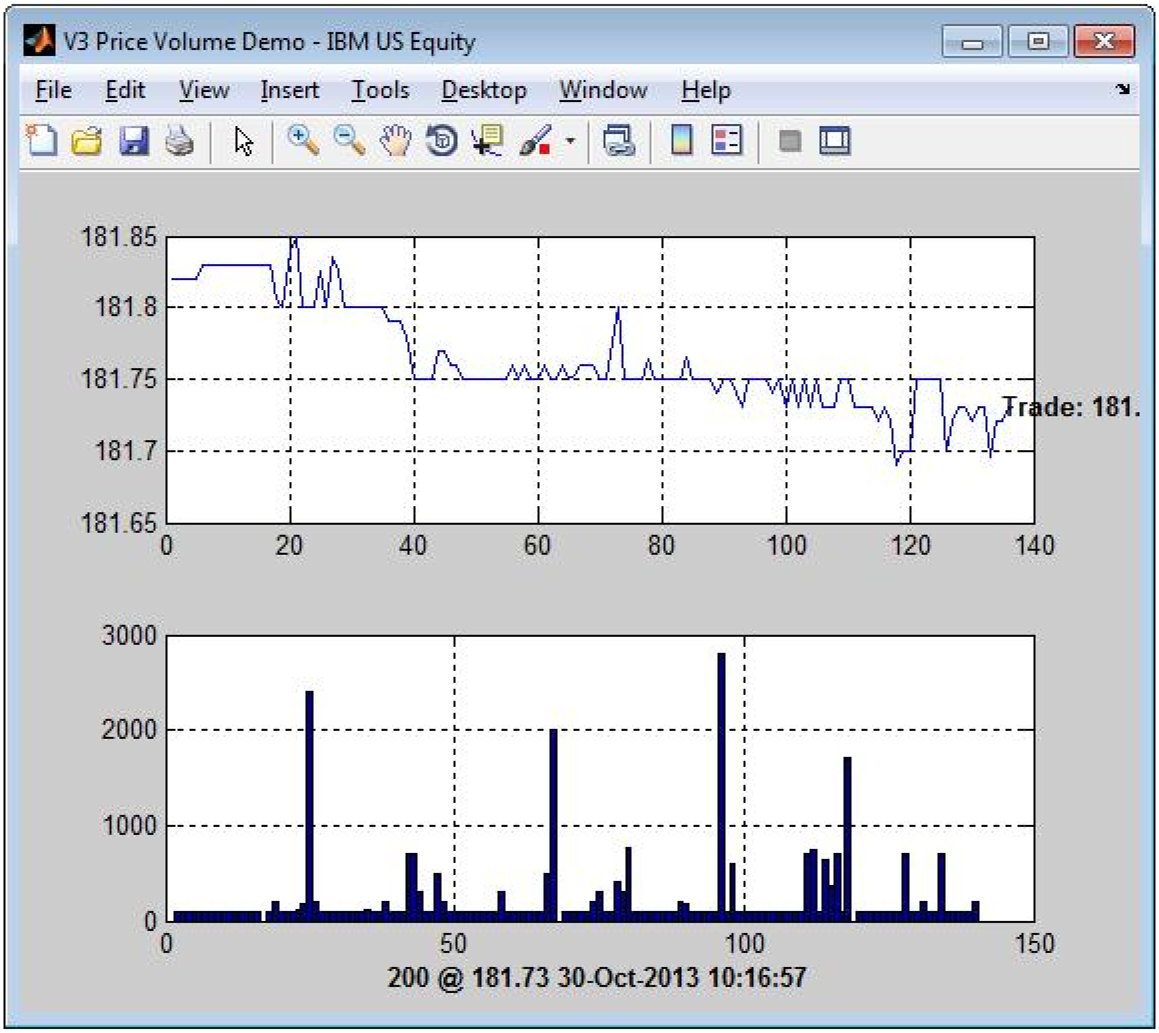
Alternatively, you can connect to the Bloomberg Server API using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

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

`v3pricevol` requires the input argument `f` of `realtime` to be `'Last_Price'` , `'Volume'`, or both.

```
[subs,t] = realtime(c,'IBM US Equity',{'Last_Price','Volume'},...
                    'v3pricevol')

subs =

com.bloomberglp.blpapi.SubscriptionList@16f66676


   Timer Object: timer-5

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

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

`realtime` returns the Bloomberg subscription list object `subs` and the MATLAB timer object with its properties. Then, `v3pricevol` displays a figure showing last price and volume data for IBM.


V3 Price Volume Demo - IBM US Equity
File Edit View Insert Tools Desktop Window Help
181.85
181.8
181.75
181.7
181.65
0 20 40 60 80 100 120 140
Trade: 181.
3000
2000
1000
0
0 50 100 150
200 @ 181.73 30-Oct-2013 10:16:57

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

Close the Bloomberg connection.

```
close(c)
```

- "Retrieve Bloomberg Real-Time Data"

Input Arguments

`c` — Bloomberg connection

connection object

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

`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 string specific to Bloomberg for one data field or a cell array of strings specific to Bloomberg 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`.

More About

- "Workflow for Bloomberg"
- "Writing and Running Custom Event Handler Functions" on page 1-29

See Also

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

stop

Unsubscribe real-time requests for Bloomberg connection V3

Syntax

```
stop(c,subs,t)
stop(c,subs,[],s)
```

Description

`stop(c,subs,t)` unsubscribes real-time requests associated with the Bloomberg connection `c` and subscription list `subs`. `t` is the timer associated with the real-time callback for the subscription list.

`stop(c,subs,[],s)` unsubscribes real-time requests for each security `s` on the subscription list `subs`. The timer input is empty.

Examples

Stop Real-Time Requests

Unsubscribe to real-time data for one security.

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the last trade and volume for the IBM security using the event handler `v3stockticker`.

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

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

```
** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

`realtime` returns the stock tick data for the IBM security with the volume and last trade price.

Stop the real-time data requests for the IBM security using the Bloomberg subscription `subs` and MATLAB timer object `t`.

```
stop(c,subs,t)
```

Close the Bloomberg connection.

```
close(c)
```

Stop Real-Time Requests for a Security List

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the last trade and volume for the security list `s` using the event handler `v3stockticker`. `s` contains securities for IBM, Google, and Ford Motor Company.

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

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

```
** IBM US Equity ** 100 @ 181.81 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.795 29-Oct-2013 15:48:50
** IBM US Equity ** 100 @ 181.8065 29-Oct-2013 15:48:51
...
```

`realtime` returns the stock tick data for the securities list `s` with the volume and last trade price.

Stop the real-time data requests for the securities list `s` using the Bloomberg subscription `subs`.

```
stop(c,subs,[],s)
```

Close the Bloomberg connection.

```
close(c)
```

- “Retrieve Bloomberg Real-Time Data”

Input Arguments

`c` — Bloomberg connection
connection object

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

`subs` — Bloomberg subscription
object

Bloomberg subscription, specified as a Bloomberg object. For details about this object, see the *Bloomberg API Developer’s Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

`t` — MATLAB timer
object

MATLAB timer, specified as a MATLAB object. For details about this object, see `timer`.

`s` — Security list
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`

More About

- “Workflow for Bloomberg”

See Also

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

tahistory

Return historical technical analysis from Bloomberg connection V3

Syntax

```
d = tahistory(c)
d = tahistory(c,s,startdate,enddate,study,period,Name,Value)
```

Description

`d = tahistory(c)` returns the Bloomberg V3 session technical analysis data study and element definitions.

`d = tahistory(c,s,startdate,enddate,study,period,Name,Value)` returns the Bloomberg V3 session technical analysis data study and element definitions with additional options specified by one or more Name,Value pair arguments.

Examples

Request 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;
```

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

List the available Bloomberg studies.

```
d = tahistory(c)
```

```
d =
         dmiStudyAttributes: [1x1 struct]
       smavgStudyAttributes: [1x1 struct]
        bollStudyAttributes: [1x1 struct]
         maoStudyAttributes: [1x1 struct]
          fgStudyAttributes: [1x1 struct]
         rsiStudyAttributes: [1x1 struct]
        macdStudyAttributes: [1x1 struct]
         tasStudyAttributes: [1x1 struct]
       emavgStudyAttributes: [1x1 struct]
      maxminStudyAttributes: [1x1 struct]
        ptpsStudyAttributes: [1x1 struct]
        cmciStudyAttributes: [1x1 struct]
        wlprStudyAttributes: [1x1 struct]
       wmavgStudyAttributes: [1x1 struct]
     trenderStudyAttributes: [1x1 struct]
         gocStudyAttributes: [1x1 struct]
        kltnStudyAttributes: [1x1 struct]
    momentumStudyAttributes: [1x1 struct]
         rocStudyAttributes: [1x1 struct]
         maeStudyAttributes: [1x1 struct]
       hurstStudyAttributes: [1x1 struct]
        chkoStudyAttributes: [1x1 struct]
          teStudyAttributes: [1x1 struct]
       vmavgStudyAttributes: [1x1 struct]
       tmavgStudyAttributes: [1x1 struct]
         atrStudyAttributes: [1x1 struct]
         rexStudyAttributes: [1x1 struct]
         adoStudyAttributes: [1x1 struct]
          alStudyAttributes: [1x1 struct]
         etdStudyAttributes: [1x1 struct]
         vatStudyAttributes: [1x1 struct]
        tvatStudyAttributes: [1x1 struct]
          pdStudyAttributes: [1x1 struct]
          rvStudyAttributes: [1x1 struct]
      ipmavgStudyAttributes: [1x1 struct]
       pivotStudyAttributes: [1x1 struct]
          orStudyAttributes: [1x1 struct]
         pcrStudyAttributes: [1x1 struct]
          bsStudyAttributes: [1x1 struct]
```

`d` contains structures pertaining to each available Bloomberg study.

Display the name-value pairs for the DMI study.

```
d.dmiStudyAttributes
```

```
ans = 

              period: [1x104 char]
     priceSourceHigh: [1x123 char]
      priceSourceLow: [1x121 char]
    priceSourceClose: [1x125 char]
```

Obtain more information about the `period` property.

```
d.dmiStudyAttributes.period
```

```
ans =

DEFINITION period {

    Min Value = 1

    Max Value = 1

    TYPE Int64

} // End Definition: period
```

Run the DMI study for the IBM security for the last month with `period` equal to `14`, the high price, the low price, and the closing price.

```
d = tahistory(c,'IBM US Equity',floor(now)-30,floor(now),'dmi',...
             'all_calendar_days','period',14,...
             'priceSourceHigh','PX_HIGH',...
             'priceSourceLow','PX_LOW','priceSourceClose','PX_LAST')
```

```
d = 

         date: [31x1 double]
     DMI_PLUS: [31x1 double]
    DMI_MINUS: [31x1 double]
          ADX: [31x1 double]
         ADXR: [31x1 double]
```

`d` contains a `studyDataTable` with one `studyDataRow` for each interval returned.

Display the first five dates in the returned data.

```
d.date(1:5,1)
```

```
ans =
      735507.00
      735508.00
      735509.00
      735510.00
      735511.00
```

Display the first five prices in the plus DI line.

```
d.DMI_PLUS(1:5,1)
```

```
ans =

         18.92
         17.84
         16.83
         15.86
         15.63
```

Display the first five prices in the minus DI line.

```
d.DMI_MINUS(1:5,1)
```

```
ans =

         30.88
         29.12
         28.16
         30.67
         29.24
```

Display the first five values of the Average Directional Index.

```
d.ADX(1:5,1)
```

```
ans =

         22.15
         22.28
         22.49
         23.15
         23.67
```

Display the first five values of the Average Directional Movement Index Rating.

```
d.ADXR(1:5,1)
```

```
ans =

         25.20
         25.06
         25.05
         25.60
         26.30
```

Close the Bloomberg connection.

```
close(c)
```

Request 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;
```

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

Run the DMI study for the Microsoft security with pricing source `ETPX` for the last month with `period` equal to `14`, the high price, the low price, and the closing price.

```
d = tahistory(c,'MSFT@ETPX US Equity',floor(now)-30,floor(now),...
             'dmi','all_calendar_days','period',14,...
             'priceSourceHigh','PX_HIGH','priceSourceLow','PX_LOW',...
             'priceSourceClose','PX_LAST')
```

```
d =

         date: [31x1 double]
     DMI_PLUS: [31x1 double]
    DMI_MINUS: [31x1 double]
          ADX: [31x1 double]
         ADXR: [31x1 double]
```

`d` contains a `studyDataTable` with one `studyDataRow` for each interval returned.

Display the first five dates in the returned data.

```
d.date(1:5,1)
```

```
ans =

     735507.00
     735508.00
     735509.00
     735510.00
     735511.00
```

Display the first five prices in the plus DI line.

```
d.DMI_PLUS(1:5,1)
```

```
ans =

         28.37
         30.63
         32.72
         30.65
         29.37
```

Display the first five prices in the minus DI line.

```
d.DMI_MINUS(1:5,1)
```

```
ans =

         21.97
         21.17
         19.47
         18.24
         17.48
```

Display the first values of the Average Directional Index.

```
d.ADX(1:5,1)
```

```
ans =

         13.53
         13.86
         14.69
         15.45
         16.16
```

Display the first five values of the Average Directional Movement Index Rating.

```
d.ADXR(1:5,1)

ans =

         15.45
         15.36
         15.53
         15.85
         16.37
```

Close the Bloomberg connection.

```
close(c)
```

Input Arguments

`c` — Bloomberg connection
connection object

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

`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`

`study` — Study type
string

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

Data Types: `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'}`, this function returns daily data for all calendar days reporting missing data as `NaN`s. When `period` is set to `{'actual'}`, this 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. This table shows the values for `period`.

Value	Description
`'daily'`	Return data for each day.
`'weekly'`	Return data for each week.
`'monthly'`	Return data for each month.
`'quarterly'`	Return data for each quarter.
`'semi_annually'`	Return data semiannually.
`'yearly'`	Return data for each year.
`'actual'`	Anchor date specification for an actual date. The anchor date is the date to which all other reported dates are related. For this function, for periodicities other than daily, `enddate` is the anchor date. For example, if you set the period to weekly and the `enddate` is a Thursday, every reported data point would also be a

Value	Description
	Thursday, or the nearest prior business day to Thursday. Similarly, if you set the period to monthly and the `enddate` is the 20th of a month, each reported data point would be for the 20th of each month in the date range.
`'calendar'`	Anchor date specification for a calendar year.
`'fiscal'`	Anchor date specification for a fiscal year.
`'non_trading_weekdays'`	Return data for all weekdays.
`'all_calendar_days'`	Return data for all calendar days.
`'active_days_only'`	Return data for only active trading days.
`'previous_value'`	Fill missing values with previous values for dates without trading activity for the security.
`'nil_value'`	Fill missing values with a `NaN` for dates without trading activity for the security.

Data Types: `char` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside 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.

More About

- "Workflow for Bloomberg"

See Also

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

timeseries

Intraday tick data for Bloomberg connection V3

Syntax

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

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

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(c,s,date,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(c,s,{startdate,enddate},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(c,s,{startdate,enddate},[],field,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`.

Examples

Retrieve Time-Series Tick Data for a Specific Date

Create the Bloomberg connection.

```
c = blp;
```

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

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;
```

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

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;
```

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

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      252322.00         861.00
   735537.40        181.90         181.97         181.57         181.65       78570.00         535.00
   735537.40        181.73         182.18         181.58         182.07      124898.00         817.00
     ...

  Column 8

   45815588.00
   14282076.00
   22710954.00
```

```
...
```

The columns in `d` contain the following:

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

Here, the first row of data shows that on today's date the open price is $181.69, the high price is $181.99, the low price is $180.10, the closing 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;
```

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

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;
```

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

Retrieve the tick series for the `'F US Equity'` security for the last business day from the beginning of the day to noon.

```
d = timeseries(c,'F US Equity',{floor(now-4),floor(now-3.5)})
```

```
d =

    'TRADE'    [735552.67]    [17.09]    [    200.00]
    'TRADE'    [735552.67]    [17.09]    [    100.00]
    'TRADE'    [735552.67]    [17.09]    [    100.00]
   ...
```

`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;
```

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

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

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

```
ans =

  Columns 1 through 7

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

Column 8

   38902968.00
    8779374.00
    9626896.00
   ...
```

The columns in `d` contain the following:

- Numeric representation of date and time
- Open price
- High price
- Low price
- Closing price
- Volume of ticks
- Number of ticks

- Total tick value in the bar

The first row of data shows that on today’s date the open price is $187.20, the high price is $187.60, the low price is $187.02, the closing price is $187.08, the volume of ticks 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;
```

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

Return the Bid, Ask, and trade tick series for the security `'F US Equity'` for yesterday with a time interval at noon, without specifying the aggregation parameter.

```
d = timeseries(c,'F US Equity',{floor(now-1)+.5,floor(now-1)+.51},...
               [],{'Bid','Ask','Trade'})
```

```
d =

    'TRADE'    [735550.50]    [16.71]    [100.00]
    'ASK'      [735550.50]    [16.71]    [312.00]
    'BID'      [735550.50]    [16.70]    [177.00]
    ...
```

`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;
```

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

Return the trade tick series for the security `'F US Equity'` for yesterday with a time interval at noon, without specifying the aggregation parameter. 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)
```

- “Retrieve Bloomberg Intraday Tick Data”

Input Arguments

`c` — Bloomberg connection
connection object

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

`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.

Limitations

When the data request is too large, `timeseries` displays this error message:

```
Timeout error:
Error using blp/timeseries>processResponseEvent (line 338) REQUEST FAILED: responseErro

source = bbdbl7

code = -2

category = TIMEOUT

message = Timed out getting data from store [nid:327]

subcategory = INTERNAL_ERROR

}
```

To fix this error, shorten the length of the date range by modifying the input arguments `startdate` and `enddate`.

More About

Tips

- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/javaclasspath.txt`. For details about the static Java class path, see “Static Path”.
- You cannot retrieve Bloomberg intraday tick data for a date more than 140 days ago.
- The *Bloomberg API Developer’s Guide* states that `'TRADE'` corresponds to LAST_PRICE for IntradayTickRequest and IntradayBarRequest.
- Bloomberg V3 intraday tick data supports additional name-value pairs. For details on these pairs, see the *Bloomberg API Developer’s Guide* by typing `WAPI` and clicking the **<GO>** button on the Bloomberg terminal.

- You can check data and field availability by using the Bloomberg Excel Add-In.
- “Workflow for Bloomberg”

See Also

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

datastream

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`

datastream.close

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

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: • `'d'`: daily values • `'w'`: weekly values • `'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 Thomson Reuters Datastream connection.

See Also

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

get

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: • `user` • `datasource` • `endpoint` • `wsdl` • `sources` • `systeminfo` • `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`

isconnection

Determine if 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

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.IESignal.dll');
```

Create an eSignal connection handle:

```
% Enter 'mylogin' as your user name.
```

```
E = esig('mylogin')
```

See Also

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

close

Close eSignal connection

Syntax

```
close(e)
```

Description

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

See Also

`esig`

getdata

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` | `timeseries` | `history`

getfundamentaldata

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

history

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` | `timeseries` | `getdata`

timeseries

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.

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')
```

More About

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`.

See Also

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

iqf

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

close

Close IQFEED ports

Syntax

```
close(Q)
```

Description

`close(Q)` closes all IQFEED ports currently open for a given IQFEED connection handle, `Q`.

Arguments

`Q`	IQFEED connection handle created using `iqf.`

Examples

Close all ports for an IQFEED connection handle.

```
close(Q)
```

See Also

`iqf`

history

IQFEED asynchronous historical end-of-period data

Syntax

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

history(c,s,{startdate,enddate})
history(c,s,{startdate,enddate},[],listener,eventhandler)
```

Description

`history(c,s,interval)` returns asynchronous historical end-of-period data using the connection object `c`, a single security `s`, and a specified interval `interval`.

`history(c,s,interval,period)` returns asynchronous historical end-of-period data for a single security with a specified interval and period `period`.

`history(c,s,interval,period,listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a specified interval, period, socket listener `listener`, and event handler `eventhandler`.

`history(c,s,{startdate,enddate})` returns asynchronous historical end-of-period data for a single security with a date range.

`history(c,s,{startdate,enddate},[],listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a date range, a specified socket listener `listener`, and event handler `eventhandler`.

Examples

Retrieve Daily Data

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five days.

```
history(c,'GOOG',5)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:08:58'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033.56'    '1277772'    '0'
```

Each row of data represents one day. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Weekly Data

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five weeks.

```
history(c,'GOOG',5,'Weekly')
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:07:02'    '1048.74'    '1020.36'    '1035.75'    '1034.07'    '4949900'     '0'
    '2013-11-15 11:07:02'    '1039.75'    '1005.00'    '1009.51'    '1033.56'    '6361983'     '0'
    '2013-11-08 11:07:02'    '1032.37'    '1007.64'    '1031.50'    '1016.03'    '6209876'     '0'
    '2013-11-01 11:07:02'    '1041.52'    '1012.98'    '1015.20'    '1027.04'    '7025769'     '0'
    '2013-10-25 11:07:02'    '1040.57'    '995.79'     '1011.46'    '1015.20'    '12636223'    '0'
```

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

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Monthly Data with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five months. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data.

```
history(c,'GOOG',5,'Monthly',@iqhistoryfeedlistener,...
        @iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:13:07'    '1048.74'    '1005.00'    '1031.79'    '1034.07'    '18805697'    '0'
    '2013-10-31 11:13:07'    '1041.52'    '842.98'     '880.25'     '1030.58'    '55288774'    '0'
    '2013-09-30 11:13:07'    '905.99'     '853.95'     '854.36'     '875.91'     '33147210'    '0'
    '2013-08-30 11:13:07'    '909.71'     '845.56'     '895.00'     '846.90'     '33509358'    '0'
    '2013-07-31 11:13:07'    '928.00'     '875.61'     '886.45'     '887.75'     '51277966'    '0'
```

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

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Data for a Date Range

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve IBM security data for the last five days.

```
history(c,'IBM',{floor(now-5),floor(now)})
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =
    '2013-11-21 10:59:51'    '185.7500'    '183.4110'    '185.5400'    '184.1300'    '4459451'    '0'
    '2013-11-20 10:59:51'    '186.2400'    '184.6450'    '185.2200'    '185.1900'    '3646117'    '0'
    '2013-11-19 10:59:51'    '186.2000'    '184.1500'    '184.6300'    '185.2500'    '4577037'    '0'
    '2013-11-18 10:59:51'    '184.9900'    '183.2700'    '183.5200'    '184.4700'    '5344864'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Data for a Date Range with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five days. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data. The period `[]` specifies the default period for daily data.

```
history(c,'GOOG',{floor(now-5),floor(now)},[],...
        @iqhistoryfeedlistener,@iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:12:15'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:12:15'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:12:15'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:12:15'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

Input Arguments

`c` — IQFEED connection

connection object

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

`s` — Security

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. You can find the code for the existing listener function in the `history.m` file.

Data Types: `function_handle`

`eventhandler` — Event handler
function

Event handler, specified as a function to process the IQFEED data. The existing event handler displays the IQFEED data in the Command Window. You can modify the existing event handler function or define your own. You can find the code for the existing event handler function in the `history.m` file.

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`

More About

Tips

- When you make multiple requests with multiple messages, this error might occur:

  Warning: Error occurred while executing delegate callback: Message: The IAsyncResult object was not returned from the corresponding asynchronous method on this class.

  To fix this, restart MATLAB.
- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

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

marketdepth

IQFEED asynchronous level 2 data

Syntax

```
marketdepth(Q, S)
marketdepth(Q, S, elistener, ecallback)
```

Description

`marketdepth(Q, S)` returns asynchronous level 2 data using the default socket listener and event handler.

`marketdepth(Q, S, elistener, ecallback)` returns asynchronous level 2 data using an explicitly defined socket listener and event handler.

Arguments

`Q`	IQFEED connection handle created using `iqf.`
`S`	`S` is specified as a 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')
```

More About

- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

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

news

IQFEED asynchronous news data

Syntax

```
news(Q, S)
news(Q, S, elistener, ecallback)
```

Description

`news(Q, S)` returns asynchronous news data using the default socket listener and event handler.

`news(Q, S, elistener, ecallback)` returns asynchronous news data using an explicitly defined socket listener and event handler.

The syntax `news(Q,true)` turns on news updates for the list of currently subscribed level 1 securities and `news(Q,false)` turns off news updates for the list of currently subscribed level 1 securities.

Arguments

`Q`	IQFEED connection handle created using `iqf.`
`S`	`S` is specified as a 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')
```

More About

- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

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

realtime

IQFEED asynchronous level 1 data

Syntax

```
realtime(Q, S)
realtime(Q, S, F)
realtime(Q, S, elistener, ecallback)
```

Description

`realtime(Q, S)` returns asynchronous level 1 data using the current update field list, default socket listener, and event handler.

`realtime(Q, S, F)` returns asynchronous level 1 data for a specified field list using the default socket listener and event handler.

`realtime(Q, S, elistener, ecallback)` returns asynchronous level 1 data using an explicitly defined socket listener and event handler.

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')
```

More About

- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

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

timeseries

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')
```

More About

Tips

- When you make multiple requests with multiple messages, this error might occur: Warning: Error occurred while executing delegate callback: Message: The

IAsyncResult object was not returned from the corresponding asynchronous method on this class. To fix this, restart MATLAB.

- "Writing and Running Custom Event Handler Functions" on page 1-29

See Also

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

factset

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`

close

Close connection to FactSet

Syntax

```
close(Connect)
```

Arguments

`Connect`	FactSet connection object created with `factset`.

Description

`close(Connect)` closes the connection to FactSet data.

See Also

`factset`

fetch

Request data from FactSet

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`Library`	FactSet formula library.
`Security`	A MATLAB 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: • `'d'`: daily values • `'b'`: business day daily values

	• `'m'`: monthly values • `'mb'`: beginning monthly values • `'me'`: ending monthly values • `'q'`: quarterly values • `'qb'`: beginning quarterly values • `'qe'`: ending quarterly values • `'y'`: annual values • `'yb'`: beginning annual values • `'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`

get

Retrieve properties of FactSet connection object

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`PropertyName`	(Optional) A MATLAB string or cell array of strings containing property names. Property names are: • `user` • `serial` • `password` • `cid`

Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the FactSet connection object.

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

Examples

Establish a connection to FactSet data:

```
Connect = factset('Fast_User','1234','Fast_Pass','userid')
```

Retrieve the connection property value:

```
h = get(Connect)
h=
       user: 'Fast_User'
     serial: '1234'
   password: 'Fast_Pass'
        cid: 'userid'
```

Retrieve the value of the connection's `user` property:

```
get(Connect, 'user')
ans =
Fast_User
```

See Also

`close` | `factset` | `fetch` | `isconnection`

isconnection

Determine if connections to FactSet are valid

Syntax

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

Arguments

`Connect`	FactSet connection object created with `factset`.

Description

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

Examples

Establish a connection, `c`, to FactSet data:

```
c = factset
```

Verify that `c` is a valid connection:

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

See Also

`close` | `factset` | `fetch` | `get`

fds

Create FactSet Data Server connection

Syntax

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

Description

`c = 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.

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

Examples

Create FDS Connection

Connect to the FactSet Data Server.

```
c = 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.

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

This creates the connection object `c`.

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

`c` — FactSet Data Server connection
connection object

FactSet Data Server connection, returned as a connection object.

See Also
`close` | `realtime` | `stop`

realtime

Obtain real-time data from FactSet Data Server

Syntax

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

Description

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

`T = realtime(c,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.

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 = realtime(c,'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 = realtime(c,'FDS1','ABCD-USA')
```

The default event handler is used which returns a structure `X` to the base MATLAB workspace containing the latest data for the symbol `'ABCD-USA'`. `X` is updated as new message events are received.

Input Arguments

`c` — FactSet Data Server connection

connection object

FactSet Data Server connection, specified as a connection object created 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.

More About

- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

`close` | `fds` | `stop`

stop

Cancel real-time request

Syntax

```
stop(c,T)
```

Description

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

Examples

Cancel FactSet Data Server Real-Time Request

Terminate a FactSet Data Server real-time request.

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

Input Arguments

`c` — FactSet Data Server connection
connection object

FactSet Data Server connection, specified as a connection object created using `fds`.

`T` — Real-time request tag
nonnegative integer

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

Data Types: `double`

See Also

`close` | `fds` | `realtime`

close

Disconnect from FactSet Data Server

Syntax

```
close(c)
```

Description

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

Examples

Close FactSet Data Server Connection

Close the FactSet Data Server connection.

```
T = realtime(c,'FDS1','GOOG-USA')
close(c)
```

Input Arguments

`c` — FactSet Data Server connection
connection object

FactSet Data Server connection, specified as a connection object created using `fds`.

See Also

`fds` | `realtime` | `stop`

fred

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`

close

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

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` | `fred` | `get` | `isconnection`

get

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: • `'url'` • `'ip'` • `'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

Determine if 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` | `fred` | `get`

haver

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

Set Haver Analytics aggregation mode

Syntax

```
X = aggregation (C)
X = aggregation (C,V)
```

Description

`X = aggregation (C)` returns the current aggregation mode.

`X = aggregation (C,V)` sets the current aggregation mode to `V`. The following table lists possible values for `V`.

Value of `V`	Aggregation mode	Behavior of `aggregation` function
`0`	`strict`	`aggregation` does not fill in values for missing data.
`1`	`relaxed`	`aggregation` fills in missing data based on data available in the requested period.
`2`	`forced`	`aggregation` fills in missing data based on some past value.
`-1`	Not recognized	`aggregation` resets `V` to its last valid setting.

See Also

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

close

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

Request data from Haver Analytics database

Syntax

```
d = fetch(c,variable)
d = fetch(c,variable,startdate,enddate)
d = fetch(c,variable,startdate,enddate,period)
```

Description

`d = fetch(c,variable)` returns historical data for the Haver Analytics variable `s`, using the connection object `c`.

`d = fetch(c,variable,startdate,enddate)` returns historical data between the dates `startdate` and `enddate`.

`d = fetch(c,variable,startdate,enddate,period)` returns historical data in time periods specified by `period`.

Examples

Retrieve Variable Data

Connect to the Haver Analytics database.

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

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

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

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

Display the first three rows of data.

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

```
ans =

     715511.00          2.38
     715512.00          2.50
     715515.00          2.50
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics database connection.

```
close(c)
```

Retrieve Variable Data for a Specified Date Range

Connect to the Haver Analytics database.

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

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

```
variable = 'FFED'; % return data for FFED
startdate = '01/01/2005'; % start of date range
enddate = '12/31/2005';   % end of date range

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

Display the first three rows of data.

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

```
ans =

     732315.00          2.25
     732316.00          2.25
     732317.00          2.25
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics database connection.

```
close(c)
```

Retrieve Quarterly Data for a Specified Date Range

Connect to the Haver Analytics database.

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

Retrieve the information of the Haver Analytics variable `'FFED'`. The descriptor for this variable is Federal Funds [Effective] Rate (% p.a.).

```
variable = 'FFED';

x = info(c,variable);
```

`info` returns the structure `x` containing fields describing the Haver Analytics variable.

Retrieve quarterly data. When you specify a date that is outside the date range in the variable, you might experience unexpected results. To prevent this, use the `EndDate` field for the end of the date range.

```
startdate = '06/01/2000';  % start of date range
enddate = x.EndDate;       % end of date range
period = 'q';              % quarterly data

d = fetch(c,variable,startdate,enddate,period)
```

Display the first three rows of data.

```
d(1:3,:)

ans =

     730759.00           6.52
     730851.00           6.50
     730941.00           5.61
```

`d` contains the numeric representation of the date in the first column and the closing value in the second column.

Close the Haver Analytics database connection.

```
close(c)
```

Input Arguments

c — Haver Analytics connection

connection object

Haver Analytics connection, specified as a connection object created using `haver`.

`variable` — Haver Analytics variable

string

Haver Analytics variable, specified as a string to denote which historical data to retrieve.

Example: `'FFED'`

Data Types: `char`

`startdate` — Start date

string | MATLAB date number

Start date, specified as a string or MATLAB date number denoting the beginning of the date range to retrieve data.

Data Types: `double` | `char`

`enddate` — End date

string | MATLAB date number

End date, specified as a string or MATLAB date number denoting the end of the date range to retrieve data.

Data Types: `double` | `char`

`period` — Period

`'d'` | `'w'` | `'m'` | `'q'` | `'a'`

Period, specified as one of the following enumerated strings that denotes the time period for the historical data.

- `'d'` for daily values
- `'w'` for weekly values
- `'m'` for monthly values
- `'q'` for quarterly values
- `'a'` for annual values

Data Types: `char`

Output Arguments

`d` — Historical data

matrix

Historical data, returned as a matrix with the numeric representation of the date in the first column and the value in the second column.

See Also

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

get

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` | `haver` | `isconnection`

info

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 `'FFED2'`:

```
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` | `haver` | `isconnection` | `nextinfo`

isconnection

Determine if 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

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

Run Haver Analytics graphical user interface (GUI)

Syntax

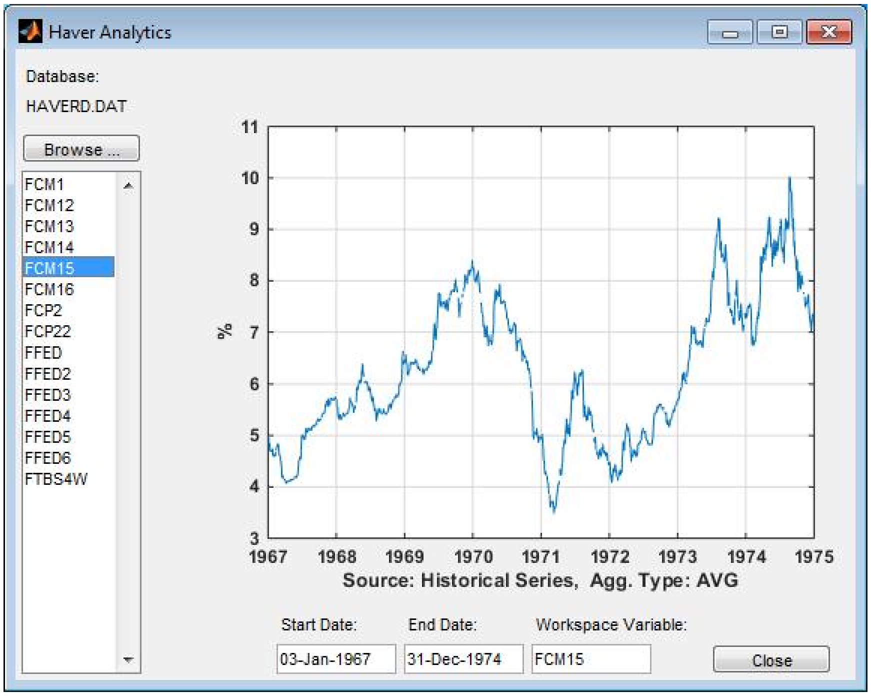
```
havertool(H)
```

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.


Haver Analytics
Database:
HAVERD.DAT
Browse ...
FCM1
FCM12
FCM13
FCM14
FCM15
FCM16
FCP2
FCP22
FFED
FFED2
FFED3
FFED4
FFED5
FFED6
FTBS4W
%
11
10
9
8
7
6
5
4
3
1967
1968
1969
1970
1971
1972
1973
1974
1975
Source: Historical Series, Agg. Type: AVG
Start Date:
End Date:
Workspace Variable:
03-Jan-1967
31-Dec-1974
FCM15
Close

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

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`

close

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

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: • `'F'` (All valid field categories) • `'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`

get

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: • `'Connected'` • `'Connection'` • `'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`

isconnection

Determine if 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`

kx

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 | fetch | get | tables

close

Close connections to Kx Systems, Inc. kdb+ databases

Syntax

```
close(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with `kx`.

Description

`close(k)` closes the connection to the Kx Systems, Inc. kdb+ database.

Examples

Close the connection, `k`, to the Kx Systems, Inc. kdb+ database:

```
close(k)
```

See Also

`kx`

exec

Run Kx Systems, Inc. kdb+ commands

Syntax

```
exec(k,command)
exec(k,command,p1,p2,p3)
exec(k,command,p1)
exec(k,command,p1,p2)
exec(k,command,p1,p2,p3)
exec(k,command,p1,p2,p3,sync)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with `kx`.
`command`	Kx Systems, Inc. kdb+ command issued using the Kx Systems, Inc. kdb + connection object created with the `kx` function.
`p1,p2,p3`	Input parameters for `Command`.

Description

`exec(k,command)` executes the specified `command` in Kx Systems, Inc. kdb+ without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the specified `command` with one or more input parameters without waiting for a response.

`exec(k,command,p1)` executes the given `command` with one input parameter without waiting for a response.

`exec(k,command,p1,p2)` executes the given `command` with two input parameters without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the given `command` with three input parameters without waiting for a response.

`exec(k,command,p1,p2,p3,sync)` executes the given `command` with three input parameters synchronously and waits for a response from the database. Enter unused parameters as empty. You can enter `sync` as 0 (default) for asynchronous commands and as 1 for synchronous commands.

Examples

Retrieve the data in the table `trade` using the connection to the Kx Systems, Inc. kdb+ database, `K`:

```
k = kx('localhost',5001);
```

Use the `exec` command to sort the data in the table `trade` in ascending order.

```
exec(k,'`date xasc`trade');
```
Subsequent data requests also sort returned data in ascending order.

After running

```
q tradedata.q -p 5001
```
at the DOS prompt, the commands

```
k = kx('localhost',5001);
exec(k,'`DATE XASC `TRADE');
```
sort the data in the table `trade` in ascending order. Data later fetched from the table will be ordered in this manner.

See Also
`fetch` | `insert` | `kx`

fetch

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 Systems, Inc. kdb+ command. `ksql` can be any valid kdb+ command. The output of the `fetch` function is any data resulting from the command specified in `ksql`.

`d = fetch(k,ksql,p1,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`

get

Retrieve Kx Systems, Inc. kdb+ connection object properties

Syntax

```
v = get(k,'PropertyName')
v = get(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with `kx`.
`'PropertyName'`	A string or cell array of strings containing property names. The property names are: • `'handle'` • `'ipaddress'` • `'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

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`

isconnection

Determine if connections to Kx Systems, Inc. kdb+ databases are valid

Syntax

```
x = isconnection(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with `kx`.

Description

`x = isconnection(k)` returns `x = 1` if the connection to the Kx Systems, Inc. kdb+ database is valid, and `x = 0` otherwise.

Examples

Establish a connection to a Kx Systems, Inc. kdb+ database, `k`:

```
k = kx('localhost',5001);
```

Verify that `k` is a valid connection:

```
x = isconnection(k)
x = 1
```

See Also

`close` | `fetch` | `get` | `kx`

tables

Retrieve table names from Kx Systems, Inc. kdb+ databases

Syntax

```
t = tables(k)
```

Arguments

`k`	The Kx Systems, Inc. kdb+ connection object created with the `kx` function.

Description

`t = tables(k)` returns the list of tables for the Kx Systems, Inc. kdb+ connection.

Examples

Retrieve table information for the Kx Systems, Inc. kdb+ database using the connection `k`:

```
t = tables(k)
t =
     'intraday'
   'seclist'
     'trade'
```

See Also

`exec` | `fetch` | `insert` | `kx`

rdth

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.TRTHApiServiceStub]
           user: 'user@company.com'
       password: '**********'
           cred: [1x1 com.thomsonreuters.tickhistory.webservice.types.CredentialsHeaderE]
    refDataFlag: 0
```

The property flag can be modified after making the connection with:

```
r.refDataFlag = true
```

or

```
r.refDataFlag = false
```

To clean up any remaining requests associated with the rdth connection use:

```
close(r)
```

See Also

`close` | `fetch` | `get`

close

Close Thomson Reuters Tick History connection

Syntax

```
close(r)
```

Description

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

See Also

`rdth`

fetch

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.

Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```
returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get information pertaining to a particular security, the command

```
d = fetch(r,'GOOG.O',{'Volume','Price','Exchange ID'},...
{'09/05/2008 12:00:00','09/05/2008 12:01:00'},...
'TimeAndSales','Trade','NSQ','EQU')
```
returns data starting with (not all data is shown):

```
d =
'#RIC'       'Date[L]'         'Time[L]'          'Type'...
     'Ex/Cntrb.ID'    'Price'
'GOOG.O'    '05-SEP-2008'    '12:00:01.178'    'Trade'...
    'NAS'            '443.86'
'Volume'
'200'
```
The command

```
d = fetch(r,'GOOG.O',{'Volume','Last'},{'09/05/2008'},...
'EndOfDay','End Of Day','NSQ','EQU')
```
returns

```
d =
    '#RIC'       'Date[L]'         'Time[L]'        ...
  'Type'    'Last'     'Volume'
    'GOOG.O'    '05-SEP-2008'    '23:59:00.000'   ...
 'End Of Day'    '444.25'    '4538375'
```
For

```
x = fetch(r,'GOOG.O')
```
for example, the exchange of the security is `x.Exchange` or `NSQ`. To determine the asset domain of the security, use the value of `x.Type`, in this case 113. Using the information from `v = get(r)`,

```
j = find(v.InstrumentTypes.Value == 113)
```
returns

```
j =46
```

The command

```
v.InstrumentTypes.Value(j)
```

returns

```
ans =
   113
```

The command

```
v.InstrumentTypes.Name(j)
```

returns

```
ans =
    'Equities'
```

The command

```
v.AssetDomains.Value(strcmp(v.InstrumentTypes.Name(j),...
v.AssetDomains.Name))
```

returns

```
ans =
    'EQU'
```

Knowing the security exchange and domain helps the interface to resolve the security symbol and return data more quickly.

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = fetch(R,'AAPL.O',{'Bid Price','Bid Size'},...
        {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

More About

Tips

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

See Also

`close` | `get` | `rdth`

get

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

`fetch` | `rdth`

isconnection

Determine if 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

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

status

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.

See Also

`close` | `rdth` | `submitftp`

submitftp

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');
```

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`

rdthloader

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

Create Reuters sessions

Syntax

```
c = reuters(session,service)
c = reuters(session,service,username,ipaddress)
c = reuters(session,service,[],[],1)
```

Description

`c = reuters(session,service)` creates a connection `c` to Reuters Market Data System (RMDS) using the Reuters session name `session` and service name `service`.

`c = reuters(session,service,username,ipaddress)` creates a Reuters connection with Data Access Control System (DACS) authentication using the user name `username` and IP address `ipaddress` of the machine running RMDS.

`c = reuters(session,service,[],[],1)` creates a Reuters connection to access only real-time data from RMDS.

Examples

Connect to RMDS

Connect to RMDS with session name `'myNS::remoteSession'` and service name `'dIDN_RDF'` without DACS authentication.

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

```
c =

  reuters with properties:

                         session: [1x1 com.reuters.rfa.internal.session.SessionImpl
                            user: []
                        position: []
                     application: '182'
```

```
                          standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIden
                              client: [1x1 com.mathworks.toolbox.datafeed.MatlabReuters
                         serviceName: 'dIDN_RDF'
                          eventQueue: [90 com.reuters.rfa.internal.common.EventQueueImp
                marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDa
    marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriber
                     mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                               defDb: [369 com.reuters.ts1.TS1DefDb]
```

`reuters` returns a Reuters connection object `c` with these properties.

- Reuters session object
- User identifier
- DACS position
- MATLAB application identifier
- Reuters Standard Principle Identity object
- Reuters client object
- Service name for connecting to the data server
- Event queue object
- Event source object
- Event source interest specification object
- Handle for event stream
- Historical data field list

Close the Reuters connection.

```
close(c)
```

Connect to RMDS Using DACS Authentication

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

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

c = 

  reuters with properties:
```

```
                          session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                             user: 'mw335'
                         position: '111.222.333.444/net'
                      application: '182'
                       standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity
                           client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClie
                      serviceName: 'dIDN_RDF'
                       eventQueue: [0 com.reuters.rfa.internal.common.EventQueueImpl]
             marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSu
marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInte
                  mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                            defDb: []
```

`reuters` returns a Reuters connection object `c` with these properties.

- Reuters session object
- User identifier
- DACS position
- MATLAB application identifier
- Reuters Standard Principle Identity object
- Reuters client object
- Service name for connecting to the data server
- Event queue object
- Event source object
- Event source interest specification object
- Handle for event stream
- Historical data field list

Close the Reuters connection.

```
close(c)
```

Connect to RMDS for Only Real-Time Data

Connect to RMDS with session name `'myNS::remoteSession'` and service name `'IDN_SELECTFEED'`. Leave user name and DACS position blank. Specify the last argument equal to `1` to retrieve only real-time data.

```
c = reuters ('myNS::remoteSession','IDN_SELECTFEED',[],[],1)
```

```
c =
  reuters with properties:

                          session: [1x1 com.reuters.rfa.internal.session.SessionImpl]
                             user: []
                         position: []
                      application: '182'
                       standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIdentity
                           client: [1x1 com.mathworks.toolbox.datafeed.MatlabReutersClie
                      serviceName: 'IDN_SELECTFEED'
                       eventQueue: [1 com.reuters.rfa.internal.common.EventQueueImpl]
             marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDataSu
marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriberInte
                  mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                            defDb: []
```

`reuters` returns a Reuters connection object `c` with these properties.

- Reuters session object
- User identifier
- DACS position
- MATLAB application identifier
- Reuters Standard Principle Identity object
- Reuters client object
- Service name for connecting to the data server
- Event queue object
- Event source object
- Event source interest specification object
- Handle for event stream
- Historical data field list

Close the Reuters connection.

```
close(c)
```

Connect to RMDS Using RTIC (TIC-RMDS Edition)

Connect to RMDS using an RTIC (TIC-RMDS Edition) connection without DACS authentication with session name `'myNS::remoteRTICSession'` and service name `'IDN_RDF'`.

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

c = 

  reuters with properties:

                           session: [1x1 com.reuters.rfa.internal.session.SessionImpl
                              user: []
                          position: []
                       application: '182'
                        standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIden
                            client: [1x1 com.mathworks.toolbox.datafeed.MatlabReuters
                       serviceName: 'IDN_RDF'
                        eventQueue: [0 com.reuters.rfa.internal.common.EventQueueImpl
              marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDa
  marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriber
                   mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                             defDb: []
```

`reuters` returns a Reuters connection object `c` with these properties.

- Reuters session object
- User identifier
- DACS position
- MATLAB application identifier
- Reuters Standard Principle Identity object
- Reuters client object
- Service name for connecting to the data server
- Event queue object
- Event source object
- Event source interest specification object
- Handle for event stream
- Historical data field list

Close the Reuters connection.

```
close(c)
```

Connect to RMDS Using RTIC (TIC-RMDS Edition) Using DACS Authentication

Connect to RMDS using an RTIC (TIC-RMDS Edition) connection with DACS authentication with:

- Session name `'myNS::remoteRTICWithDACs'`
- Service name `'IDN_RDF'`
- User name `'ab123'`
- Data server IP address `'111.222.333.444/net'`

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

```
c =

  reuters with properties:

                           session: [1x1 com.reuters.rfa.internal.session.SessionImpl
                              user: 'mw427'
                          position: '192.168.107.130'
                       application: '182'
                        standardPI: [1x1 com.reuters.rfa.common.StandardPrincipalIden
                            client: [1x1 com.mathworks.toolbox.datafeed.MatlabReuters
                       serviceName: 'IDN_RDF'
                        eventQueue: [2 com.reuters.rfa.internal.common.EventQueueImpl
              marketDataSubscriber: [1x1 com.reuters.rfa.internal.session.md.MarketDa
  marketDataSubscriberInterestSpec: [1x1 com.reuters.rfa.session.MarketDataSubscriber
                   mdsClientHandle: [1x1 com.reuters.rfa.internal.common.HandleImpl]
                             defDb: []
```

`reuters` returns a Reuters connection object `c` with these properties.

- Reuters session object
- User identifier
- DACS position
- MATLAB application identifier
- Reuters Standard Principle Identity object
- Reuters client object
- Service name for connecting to the data server

- Event queue object
- Event source object
- Event source interest specification object
- Handle for event stream
- Historical data field list

Close the Reuters connection.

```
close(c)
```

Input Arguments

`session` — Session name

string

Session name, specified as a string to denote a Reuters session.

Data Types: `char`

`service` — Service name

string

Service name, specified as a string to denote the service for connecting to the Reuters data server.

Data Types: `char`

`username` — User name

string

User name, specified as a string to denote your Reuters user identification.

Data Types: `char`

`ipaddress` — IP address

string

IP address, specified as a string to identify the machine running the Reuters data server.

Data Types: `char`

Output Arguments

c — Reuters connection
connection object

Reuters connection, returned as a Reuters connection object with these properties.

Property	Description
`session`	Reuters session object
`user`	User identifier
`position`	DACS position
`application`	MATLAB application identifier
`standardPI`	Reuters Standard Principle Identity object
`client`	Reuters client object
`serviceName`	Service name for connecting to the data server
`eventQueue`	Event queue object
`marketDataSubscriber`	Event source object
`marketDataSubscriberInterestSpec`	Event source interest specification object
`mdsClientHandle`	Handle for event stream
`defDb`	Historical data field list

More About

Tips

- Before using this function, you must configure your environment for connecting to a Reuters data server. For details, see "Reuters Configurations" on page 1-5.
- You can connect to the Reuters data server without DACS authentication. For example, use this code.

  ```
  c = reuters('myNS::remoteSession','IDN_CONFLATED');
  ```
- When you connect to RMDS without DACS authentication, ignore these informational messages that can appear in the Command Window.

```
Oct 5, 2007 2:28:31 PM
com.reuters.rfa.internal.connection.
ConnectionImpl initializeEntitlements
INFO: com.reuters.rfa.connection.ssl....
   myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

- When you connect to RMDS with DACS authentication, ignore these informational messages that can appear in the Command Window.

```
Oct 5, 2007 2:27:14 PM ...
com.reuters.rfa.internal.connection.
ConnectionImpl$ConnectionEstablishmentThread runImpl
INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
Connection successful: ...
    componentName :myNS::RTICwithDacs,
subscriberRVConnection:
{service: 9453, network: 192.168.107.0;225.2.2.8,
daemon: tcp:192.168.107.131:9450}
Oct 5, 2007 2:27:14 PM
com.reuters.rfa.internal.connection.sass3....
    Sass3LoggerProxy log
INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
SASS3JNI: Received advisory from RV session@
(9453,192.168.107.0;225.2.2.8,tcp:192.168.107.131:9450):
 _RV.INFO.SYSTEM.RVD.CONNECTED
Oct 5, 2007 2:27:14 PM
com.reuters.rfa.internal.connection.ConnectionImpl
makeServiceInfo
WARNING: com.reuters.rfa.connection.sass3....
    myNS.RTICwithDacs
Service list configuration has no
    alias defined for network
serviceName IDN_RDF
```

- “Reuters Configurations” on page 1-5

See Also

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

addric

Create Reuters Instrument Code

Syntax

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

Description

`addric(c,ric,fid,fval,type)` creates a Reuters Instrument Code, `ric`, on the service defined by the Reuters session, `c`. Supply the field ID or name, `fid`, and the field value, `fval`. Specify whether the RIC `type` is `'live'` or `'static'` (default).

Examples

Create a live RIC called `myric` with the fields `'trdprc_1'` (field ID 6) and `'bid'` (field ID 22) set to initial values of `0`:

```
addric(c,'myric',{trdprc_1','bid'},{0,0},'live')
```

Create a live RIC called `myric` with the fields `trdprc_1` and `bid` set to initial values of `0`:

```
addric(c,'myric',{6,22},{0,0},'live')
```

See Also

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

close

Release connections to Reuters data servers

Syntax

```
close(c)
```

Arguments

`c`	Reuters session object created with `reuters`.

Description

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

Examples

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

```
close(c)
```

See Also

`reuters`

contrib

Contribute data to Reuters data feed

Syntax

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

Description

`contrib(c,s,fid,fval)` contributes data to a Reuters data feed. `c` is the Reuters session object, and `s` is the RIC. Supply the field IDs or names, `fid`, and field values, `fval`.

Examples

Contribute data to the Reuters datafeed for the Reuters session object `c` and the RIC `'myric'`. Provide a last trade price of 33.5.

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

Contribute an additional bid price of 33.8:

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

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

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

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

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

See Also

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

deleteric

Delete Reuters Instrument Code

Syntax

```
deleteric(c,ric)
deleteric(c,ric,fid)
```

Description

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

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

Examples

Delete `'myric'` and all of its fields:

```
deleteric(c,'myric')
```

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

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

See Also

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

fetch

Request data from Reuters data servers

Syntax

```
d = fetch(c,sec)
d = fetch(c,sec,[],fields)
subs = fetch(c,sec,eventhandler)
```

Description

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

`d = fetch(c,sec,[],fields)` requests the given fields `fields`, for the security `sec`, given the Reuters session object `c`.

`subs = fetch(c,sec,eventhandler)` uses the Reuters session object `c` to subscribe to the security `sec`. MATLAB runs the `eventhandler` function for each data event that occurs.

Examples

Retrieve Current Securities Data

Connect to Thomson Reuters.

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

```
Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements

INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output message specifies a successful connection to the Reuters Market Data System.

Retrieve the current data for the Google security using the Reuters session object `c`.

```
sec = 'GOOG.O';

d = fetch(c,sec)

d = 

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

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

Close the Thomson Reuters connection.

```
close(c)
```

Request Specific Fields

Connect to Thomson Reuters.

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

Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements

INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output specifies a successful connection to the Reuters Market Data System.

Request the product permissions information `'PROD_PERM'` for the Google security from Reuters.

```
sec = 'GOOG.O';
field = 'PROD_PERM';

d = fetch(c,sec,[],field)

d = 

    PROD_PERM: 74
```

Request the product permissions information `'PROD_PERM'` and the display information for the IDN terminal device `'RDNDISPLAY'` for the Google security from Reuters. Use a cell array to input these two fields to the function.

```
sec = 'GOOG.O';
fields = {'PROD_PERM','RDNDISPLAY'};

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

```
d = 

     PROD_PERM: 74
    RDNDISPLAY: 66
```

Close the Thomson Reuters connection.

```
close(c)
```

Subscribe to a Security

To subscribe to a security and process the data in real time, specify an event handler function. MATLAB runs this function each time it receives a real-time data event from Reuters.

Connect to Thomson Reuters.

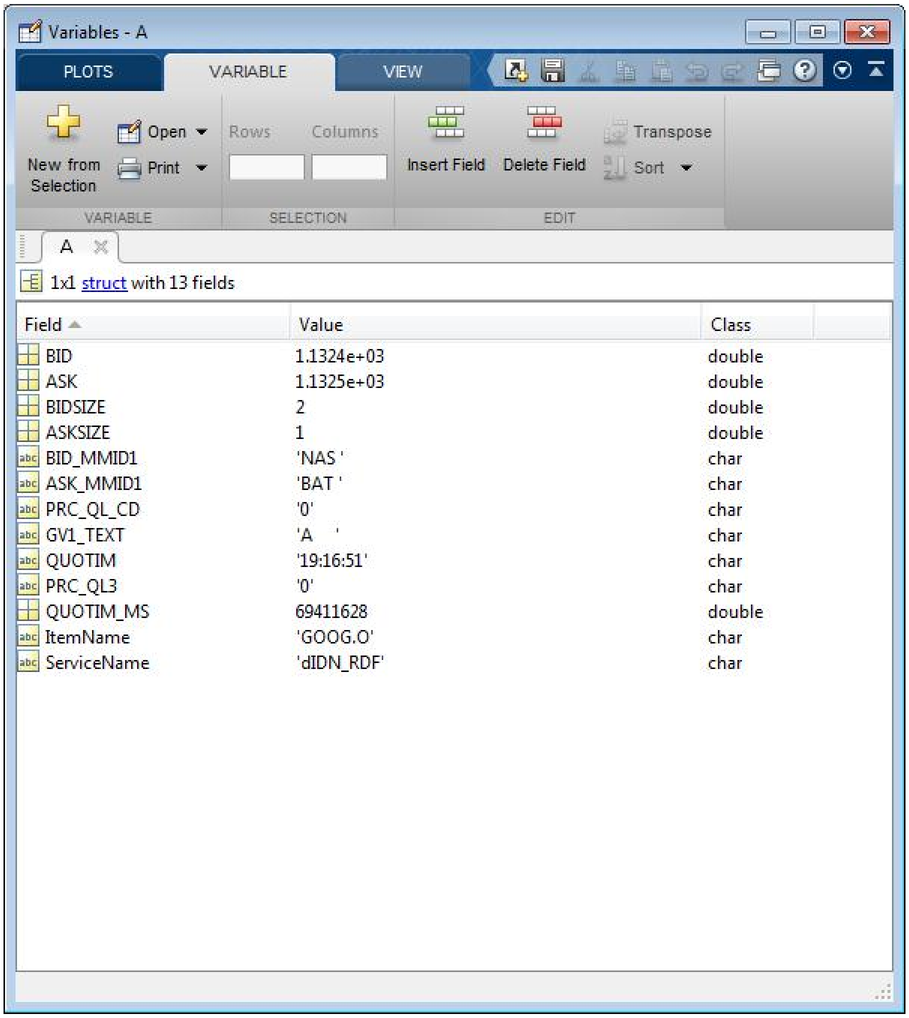
```
c = reuters('myNS::remotesession','dIDN_RDF');
```

```
Jan 13, 2014 2:23:09 PM com.reuters.rfa.internal.connection.md.MDConnectionImpl initializeEntitlements

INFO: com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

The output specifies a successful connection to the Reuters Market Data System.

The event handler `rtdemo` function returns the real-time Reuters data for the Google security to the MATLAB workspace variable `A`. `openvar` displays `A` in the Variables editor.

```
sec = 'GOOG.O';
eventhandler = 'rtdemo';

subs = fetch(c,sec,eventhandler);
openvar('A')
```



1x1 struct with 13 fields

Field	Value	Class
BID	1.1324e+03	double
ASK	1.1325e+03	double
BIDSIZE	2	double
ASKSIZE	1	double
BID_MMID1	'NAS '	char
ASK_MMID1	'BAT '	char
PRC_QL_CD	'0'	char
GV1_TEXT	'A '	char
QUOTIM	'19:16:51'	char
PRC_QL3	'0'	char
QUOTIM_MS	69411628	double
ItemName	'GOOG.O'	char
ServiceName	'dIDN_RDF'	char

In this instance, the fields represent a bid or ask tick.

The `fetch` function returns the subscription handle associated with this request in the variable `subs`. Display the subscription handle contents.

```
subs
```

```
subs =
com.reuters.rfa.internal.common.SubHandleImpl[]:
    [com.reuters.rfa.internal.common.SubHandleImpl]
```

Stop the real-time subscription.

```
stop(c,subs)
```

Close the Thomson Reuters connection.

```
close(c)
```

Input Arguments

`c` — Reuters session

object

Reuters session, specified as a Reuters session object created using `reuters`.

`sec` — Security list

string | cell array

Security list, specified as a string or a cell array of strings to denote Reuters securities.

Data Types: `char` | `cell`

`fields` — Reuters fields list

string | cell array

Reuters fields list, specified as a string or cell array of strings to denote Reuters field names.

Data Types: `char` | `cell`

`eventhandler` — Reuters real-time event handler

function

Reuters real-time event handler, specified as a MATLAB function that runs for each data event that occurs. The sample event handler called `rtdemo.m` returns Reuters real-time data from the Reuters Market Data System to the MATLAB workspace. The sample event handler specifies these input arguments.

Event Handler Input Argument	**Description**
`x`	Return data structure
`itemName`	Reuters data name
`serviceName`	Reuters service name

The sample event handler writes variable `A` to the Workspace browser with the contents of `x`.

Data Types: `function_handle`

Output Arguments

`d` — Reuters request data
structure

Reuters request data, returned as a structure. The structure contains many Reuters data fields. For details, see Reuters Data Support.

`subs` — Reuters subscription handle
object

Reuters subscription handle, returned as a Reuters subscription object.

More About

- "Writing and Running Custom Event Handler Functions" on page 1-29

See Also

`close` | `reuters` | `stop`

get

Retrieve properties of Reuters session objects

Syntax

```
e = get(c)
e = get(c,f)
```

Arguments

`c`	Reuters session object created with `reuters`.
`f`	Reuters session properties list.

Description

`e = get(c)` returns Reuters session properties for the Reuters session object `c`.

`e = get(c,f)` returns Reuters session properties specified by the properties list `f` for the Reuters session object `c`.

See Also

`reuters`

history

Request data from Reuters Time Series One

Syntax

```
d = history(c,s)
d = history(c,s,p)
d = history(c,s,f)
d = history(c,s,f,p)
d = history(c,s,d)
d = history(c,s,startdate,enddate)
d = history(c,s,startdate,enddate,p)
d = history(c,s,f,startdate,enddate)
d = history(c,s,f,startdate,enddate,p)
```

Description

`d = history(c,s)` returns all available daily historical data for the RIC, `s`, for the Reuters session object `c`.

`d = history(c,s,p)` returns all available historical data for the RIC, `s`, for the Reuters session object `c`. `p` specifies the period of the data:

- `'d'` - daily (default)
- `'w'` - weekly
- `'m'` - monthly

Note: Reuters Time Series One will only return two years of daily data, five years of weekly data, or ten years of monthly data from the current date.

`d = history(c,s,f)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`.

`d = history(c,s,f,p)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`. `p` specifies the period of the data.

`d = history(c,s,d)` returns the historical data for the RIC, `s`, for the given date, `d`, for the Reuters session object `c`.

`d = history(c,s,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,startdate,enddate,p)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

`d = history(c,s,f,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,f,startdate,enddate,p)` returns the historical data for the RIC, `s`, and fields, `f`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

Examples

`d = history(c,'WXYZ.O')` returns a structure containing all available historical end of day daily data for the RIC `'WXYZ.O'`, for the Reuters session object `c`.

`d = history(c,'WXYZ.O','close')` returns a structure with the fields `date` and `close` containing all available historical end of day daily data for the RIC `'WXYZ.O'`.

`d = history(c,'WXYZ.O','close','m')` returns all available monthly data.

`d = history(c,'WXYZ.O','01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009. Note that only two years worth of daily data, five years worth of weekly data, and 10 years of monthly data from today's date is made available by Reuters.

`d = history(c,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

`d = history(c,'WXYZ.O',`
`{'close','volume'},'01-03-2009','02-24-2009','w')` returns all available weekly data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

See Also

`close` | `fetch` | `reuters`

stop

Unsubscribe securities

Syntax

```
stop(c)
stop(c,d)
```

Arguments

`c`	Reuters session object created with `reuters`.
`d`	Subscription handle returned by `fetch`.

Description

`stop(c)` unsubscribes all securities associated with the Reuters session object `c`.

`stop(c,d)` unsubscribes the securities associated with the subscription handle `d`, where `d` is the subscription handle returned by `reuters/fetch`.

Examples

Unsubscribe securities associated with a specific request `d` and a Reuters connection object `c`:

```
stop(c,d)
```

Unsubscribe all securities associated with the Reuters connection object `c`:

```
stop(c)
```

See Also

`fetch` | `reuters`

rmdsconfig

Reuters Market Data System (RMDS) configuration editor

Syntax

```
rmdsconfig
```

Description

`rmdsconfig` opens the Reuters Market Data System configuration editor.

See Also

`reuters`

treikon

Thomson Reuters Eikon connection

Syntax

```
c = treikon
c = treikon(source)
c = treikon(source,filepath)
```

Description

`c = treikon` creates a connection to Thomson Reuters Eikon using the default data source and Thomson Reuters Eikon installation path. To create the connection, Thomson Reuters Eikon requires that you specify additional API calls.

`c = treikon(source)` creates a connection to Thomson Reuters Eikon using the data source `source` and default Thomson Reuters Eikon installation path.

`c = treikon(source,filepath)` creates a connection to Thomson Reuters Eikon using the data source `source` and the installation file path `filepath`.

Examples

Connect to Thomson Reuters Eikon

Create a Thomson Reuters Eikon connection `c` with the default data source and installation file path.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson

Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize

c =

  treikon with properties:

        Assembly: {1x6 cell}
    DataAPIClass: [1x1 EikonDesktopDataAPI.EikonDesktopDataAPIClass]
          Source: 'IDN'

ans =

Disconnected

ans =

Succeed

ans =

Connected
```

`c` contains these properties:

- References to the Microsoft .NET Framework assemblies
- Thomson Reuters Eikon `EikonDesktopDataAPI` object
- Thomson Reuters Eikon default data source

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Connect to Thomson Reuters Eikon with a Data Source

Create a Thomson Reuters Eikon connection `c` with the data source `'IDN'` and the default installation file path.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon('IDN')
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
```

```
c =

  treikon with properties:

        Assembly: {1x6 cell}
    DataAPIClass: [1x1 EikonDesktopDataAPI.EikonDesktopDataAPIClass]
          Source: 'IDN'

ans =

Disconnected

ans =

Succeed

ans =

Connected
```

`c` contains these properties:

- References to the Microsoft .NET Framework assemblies
- Thomson Reuters Eikon `EikonDesktopDataAPI` object
- Thomson Reuters Eikon data source `IDN`

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Connect to Thomson Reuters Eikon with a Data Source and File Path

Create a Thomson Reuters Eikon connection `c` with the data source `'IDN'` and the installation file path `'c:\Program Files (x86)\Thomson Reuters\Eikon \4.0.10490\Bin'`.

To return the connection status to the Command Window, use the event handler function `trestatuseventhandler` in the API method `add_OnStatusChanged`. You can modify this event handler or create your own to add other functionality. Whenever the state of the connection changes, display the Thomson Reuters Eikon status using the API property `Status`.

To initialize the Thomson Reuters Eikon Desktop, use the API method `Initialize`. This method ensures Thomson Reuters Eikon Desktop runs and connects to the Thomson Reuters Eikon Platform. To establish a successful connection, you must complete initialization successfully.

```
c = treikon('IDN',...
            'c:\Program Files (x86)\Thomson Reuters\Eikon\4.0.10490\Bin')
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
```

```
c = 

  treikon with properties:

       Assembly: {1x6 cell}
   DataAPIClass: [1x1 EikonDesktopDataAPI.EikonDesktopDataAPIClass]
         Source: 'IDN'
```

```
ans =

Disconnected

ans =

Succeed

ans =

Connected
```

`c` contains these properties:

- References to the Microsoft .NET Framework assemblies
- Thomson Reuters Eikon `EikonDesktopDataAPI` object
- Thomson Reuters Eikon data source `IDN`

When the Command Window displays the Succeed message, you have successfully initialized Thomson Reuters Eikon Desktop.

When the Command Window displays the Connected message, MATLAB connects to Thomson Reuters Eikon.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Input Arguments

`source` — Data source
string

Data source, specified as a string to denote the Thomson Reuters Eikon data source. You can configure the data source in the Thomson Reuters Eikon Desktop.

Data Types: `char`

`filepath` — File path
string

File path, specified as a string to denote the installation file path for loading the Thomson Reuters Eikon Microsoft .NET Framework DLLs. When you do not specify this input argument, the software retrieves the installation file path from the Windows registry.

Data Types: `char`

Output Arguments

`c` — Thomson Reuters Eikon connection

connection object

Thomson Reuters Eikon connection, returned as a connection object with these properties.

Property	Description
`Assembly`	Cell array of Microsoft .NET Framework assemblies loaded in MATLAB
`DataAPIClass`	Thomson Reuters Eikon `EikonDesktopDataAPI` object
`Source`	Thomson Reuters Eikon data source

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”
- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

`getdata` | `history` | `realtime` | `start` | `stop`

getdata

Retrieve current market data from Thomson Reuters Eikon

Syntax

```
d = getdata(c,s,fields)
```

Description

`d = getdata(c,s,fields)` returns the current market data from Thomson Reuters Eikon given the connection `c`, security list `s`, and fields list `fields`.

Examples

Retrieve Data for One Field

To retrieve current data, create the connection `c` using `treikon`. For an example showing this activity, see "Retrieve Thomson Reuters Eikon Current Data".

Retrieve last price data for Google.

```
s = 'GOOG.O';
field = 'LAST'; % Last price field

d = getdata(c,s,field)
```

```
ans =

RT_LIST_INACTIVE

ans =

RT_LIST_RUNNING

d =

    LAST: {[1119.58]}
```

```
ans =

GOOG.O
```

Thomson Reuters Eikon provides status messages to the Command Window. `RT_LIST_INACTIVE` means no real-time data is being requested and `RT_LIST_RUNNING` means real-time data is updating.

`getdata` returns `d` as a structure containing the field `LAST` for the last price. This field value contains the returned last price of $1119.58 for Google.

After returning the output structure, Thomson Reuters Eikon returns the contents of the security list `GOOG.O`.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Retrieve Data for Multiple Fields

To retrieve current data for multiple fields, create the connection `c` using `treikon`. For an example showing this activity, see “Retrieve Thomson Reuters Eikon Current Data”.

Retrieve last price and bid price data for Google.

```
s = 'GOOG.O';
fields = {'LAST','BID'}; % Last price and bid price fields

d = getdata(c,s,fields)
```

```
ans =

GOOG.O

d =

    LAST: {[1119.77]}
     BID: {[1119.41]}
```

`getdata` returns `d` as a structure containing the field `LAST` with the last price $1119.77 and the field `BID` with the bid price $1119.41 for Google.

To close the Thomson Reuters Eikon connection, exit MATLAB.

- “Retrieve Thomson Reuters Eikon Current Data”

Input Arguments

`c` — Thomson Reuters Eikon connection
connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

`s` — Security list
string | cell array

Security list, specified as a string for one security or a cell array for multiple securities.

Data Types: `char` | `cell`

`fields` — Requested fields list
string | cell array

Requested fields list, specified as a string for one field or a cell array for multiple fields.

Data Types: `char` | `cell`

Output Arguments

`d` — Return data
structure

Return data, returned as a structure containing fields with Thomson Reuters Eikon current market data.

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”

See Also

`history` | `realtime` | `treikon`

history

Retrieve historical data from Thomson Reuters Eikon

Syntax

```
d = history(c,s,fields)
d = history(c,s,fields,startdate,enddate)
d = history(c,s,fields,startdate,enddate,period)
```

Description

`d = history(c,s,fields)` returns historical data for a default date range for the securities `s` and the fields `fields` given the Thomson Reuters Eikon connection object `c`.

`d = history(c,s,fields,startdate,enddate)` returns historical data for a date range beginning with `startdate` and ending with `enddate`.

`d = history(c,s,fields,startdate,enddate,period)` returns historical data for a date range beginning with `startdate`, ending with `enddate`, and using periodicity `period`.

Examples

Retrieve Historical Data

To retrieve historical data, create the connection `c` using `treikon`. For an example showing this activity, see “Retrieve Thomson Reuters Eikon Historical Data”.

Retrieve the daily open, high, low, and close prices for Apple.

```
s = 'AAPL.O';
fields = {'DATE','OPEN','HIGH','LOW','CLOSE'};

d = history(c,s,fields)
```

```
d =
    '4/7/2014 12:00:0...'    [528.02]    [530.90]    [521.89]    [523.47]
    '4/4/2014 12:00:0...'    [539.81]    [540.00]    [530.58]    [531.82]
    '4/3/2014 12:00:0...'    [541.39]    [542.50]    [537.64]    [538.79]
    ...
```

`d` is a cell array that contains five columns:

- Date and time
- Open price
- High price
- Low price
- Close price

Each row represents one day of data.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Retrieve Historical Data with a Date Range

To retrieve historical data with a date range, create the connection `c` using `treikon`. For an example showing this activity, see “Retrieve Thomson Reuters Eikon Historical Data”.

Retrieve the daily open, high, low, and close prices for Apple. Retrieve data for the last 30 days.

```
s = 'AAPL.O';
fields = {'DATE','OPEN','HIGH','LOW','CLOSE'};
startdate = floor(now)-30; % Beginning of date range as of 30 days ago
enddate = floor(now); % End of date range as of today

d = history(c,s,fields,startdate,enddate)
```

```
d =

    '4/8/2014 12:00:0...'    [525.19]    [526.12]    [518.70]    [523.44]
    '4/7/2014 12:00:0...'    [528.02]    [530.90]    [521.89]    [523.47]
    '4/4/2014 12:00:0...'    [539.81]    [540.00]    [530.58]    [531.82]
    ...
```

`d` is a cell array that contains five columns:

- Date and time
- Open price
- High price
- Low price

- Close price

Each row represents one day of data. The total number of rows equals the number of trading days in the last month.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Retrieve Historical Data with a Periodicity

To retrieve historical data with a date range and periodicity, create the connection `c` using `treikon`. For an example showing this activity, see "Retrieve Thomson Reuters Eikon Historical Data".

Retrieve the weekly open, high, low, and close prices for Apple. Retrieve data for the last 30 days.

```
s = 'AAPL.O';
fields = {'DATE','OPEN','HIGH','LOW','CLOSE'};
startdate = floor(now)-30; % Beginning of date range as of 30 days ago
enddate = floor(now); % End of date range as of today
period = 'W'; % Weekly periodicity

d = history(c,s,fields,startdate,enddate,period)

d =

    '4/4/2014 12:00:0...'    [539.23]    [543.48]    [530.58]    [531.82]
    '3/28/2014 12:00:...'    [538.42]    [549.00]    [534.25]    [536.86]
    '3/21/2014 12:00:...'    [527.70]    [536.24]    [525.20]    [532.87]
    '3/14/2014 12:00:...'    [528.36]    [539.66]    [523.00]    [524.69]
```

`d` is a cell array that contains five columns:

- Date and time
- Open price
- High price
- Low price
- Close price

Each row represents one week of data. The total number of rows equals the number of weeks in the requested date range.

To close the Thomson Reuters Eikon connection, exit MATLAB.

- "Retrieve Thomson Reuters Eikon Historical Data"

Input Arguments

`c` — Thomson Reuters Eikon connection

connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

`s` — Security symbol

string | cell array

Security symbol, specified as a string or cell array containing only one symbol for a security.

Example: `{'GOOG.O'}`

Data Types: `char` | `cell`

`fields` — Requested fields list

string | cell array

Requested fields list, specified as a string for one field or a cell array for multiple fields.

Data Types: `char` | `cell`

`startdate` — Start date

scalar | string

Start date, specified as a scalar or string to denote the beginning of the date range to return historical data.

Example: `floor(now)-30`

Data Types: `double` | `char`

`enddate` — End date

scalar | string

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

Example: `floor(now)`

Data Types: `double` | `char`

`period` — Periodicity
`'D'` (default) | `'W'` | `'M'`

Periodicity, specified as one of these enumerated strings to denote the frequency of the returned historical data.

Enumerated String	Description
`'D'`	Daily
`'W'`	Weekly
`'M'`	Monthly

Data Types: `char`

Output Arguments

`d` — Return data
cell array

Return data, returned as a cell array containing Thomson Reuters Eikon historical data.

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”

See Also
`getdata` | `realtime` | `treikon`

realtime

Retrieve real-time data from Thomson Reuters Eikon

Syntax

```
subs = realtime(c,s,fields,eventhandler)
```

Description

`subs = realtime(c,s,fields,eventhandler)` subscribes to a security `s` and asynchronously returns data for the request fields `fields` using the event handler `eventhandler` to process Thomson Reuters Eikon data events.

Examples

Retrieve Real-Time Data for One Security

To retrieve real-time data, create the connection `c` using `treikon`. For an example showing this activity, see "Retrieve Thomson Reuters Eikon Real-Time Data".

Retrieve real-time data for the last price and bid price for Google. The sample event handler `trerealtimeeventhandler` retrieves the real-time data to put into the MATLAB variable `trRealtimeData` in the Workspace browser.

```
s = 'GOOG.O';
fields = {'LAST','BID'};

subs = realtime(c,s,fields,...
                @(varargin)trerealtimeeventhandler(varargin{:}))

subs =

    AdxRtListCOMObj: [1x1 System.__ComObject]
       AdxRtListObj: [1x1 ThomsonReuters.Interop.RTX.AdxRtListClass]
              Items: {'GOOG.O'}
             Fields: {'LAST'  'BID'}
         UpdateMode: [1x1 ThomsonReuters.Interop.RTX.RT_RunMode]
```

`subs` is a subscription structure that contains the security list in the structure field `Items`. `subs` contains the Thomson Reuters Eikon field list in the structure field `Fields`.

Display the real-time data for Google by accessing the contents of the variable `trRealtimeData` in the Workspace browser.

```
trRealtimeData
```

```
trRealtimeData = 

     RIC: 'GOOG.O'
    LAST: 561.26
     BID: 561.16
```

The variable `trRealtimeData` is a structure that contains real-time data. `trRealtimeData` contains the Thomson Reuters Eikon Reuters Instrument Code (RIC) in the structure field `RIC`. This structure contains any requested Thomson Reuters Eikon fields as structure fields. For example, `trRealtimeData` contains the last price of $561.26 for Google in the structure field `LAST`.

To close the Thomson Reuters Eikon connection, exit MATLAB.

Retrieve Real-Time Data for Multiple Securities

To retrieve real-time data, create the connection `c` using `treikon`. For an example showing this activity, see "Retrieve Thomson Reuters Eikon Real-Time Data".

Retrieve real-time data for last price for Google and Microsoft. The sample event handler `trerealtimeeventhandler` retrieves the real-time data to put into the MATLAB variable `trRealtimeData` in the Workspace browser.

```
s = {'GOOG.O','MSFT.O'};
fields = 'LAST';

subs = realtime(c,s,fields,...
                @(varargin)trerealtimeeventhandler(varargin{:}))
```

```
subs = 

    AdxRtListCOMObj: [1x1 System.__ComObject]
       AdxRtListObj: [1x1 ThomsonReuters.Interop.RTX.AdxRtListClass]
              Items: {'GOOG.O'  'MSFT.O'}
             Fields: {'LAST'}
```

```
        UpdateMode: [1x1 ThomsonReuters.Interop.RTX.RT_RunMode]
```

`subs` is a subscription structure that contains the security list in the structure field `Items`. `subs` contains the Thomson Reuters Eikon field list in the structure field `Fields`.

If the security list contains multiple securities, the sample event handler `trerealtimeeventhandler` retrieves real-time data for the security with the latest event. The first event occurs for Google. Display the real-time data for Google by accessing the contents of the variable `trRealtimeData` in the Workspace browser.

```
trRealtimeData

trRealtimeData = 

     RIC: 'GOOG.O'
    LAST: 564.09
```

The variable `trRealtimeData` is a structure that contains real-time data. `trRealtimeData` contains the Thomson Reuters Eikon RIC in the structure field `RIC`. This structure contains any requested Thomson Reuters Eikon fields as structure fields. For example, `trRealtimeData` contains the last price of $564.09 for Google in the structure field `LAST`.

The next event occurs for Microsoft and the variable `trRealtimeData` reflects the last price. Display the real-time data for Microsoft by accessing the contents of the variable `trRealtimeData` in the Workspace browser.

```
trRealtimeData

trRealtimeData = 

     RIC: 'MSFT.O'
    LAST: 40.55
```

To close the Thomson Reuters Eikon connection, exit MATLAB.

- "Retrieve Thomson Reuters Eikon Real-Time Data"

Input Arguments

c — Thomson Reuters Eikon connection

connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

`s` — Security list
string | cell array

Security list, specified as a string for one security or a cell array for multiple securities.

Data Types: `char` | `cell`

`fields` — Requested fields list
string | cell array

Requested fields list, specified as a string for one field or a cell array for multiple fields.

Data Types: `char` | `cell`

`eventhandler` — Event handler
function

Event handler, specified as a function to process Thomson Reuters Eikon data. You can modify the existing event handler function `trerealtimeeventhandler` or define your own function to process any real-time Thomson Reuters Eikon events. `trerealtimeeventhandler` retrieves real-time data to put into the MATLAB variable `trRealtimeData` in the Workspace browser.

Data Types: `function_handle`

Output Arguments

`subs` — Thomson Reuters Eikon subscription list
structure

Thomson Reuters Eikon subscription list, returned as a structure containing these fields.

Field	Description
`AdxRtListCOMObj`	Thomson Reuters Eikon Desktop Data API COM object
`AdxRtListObj`	Thomson Reuters Eikon Desktop Data API list object

Field	Description
`Items`	Thomson Reuters Eikon Reuters Instrument Code (RIC) list specified by the input argument s
`Fields`	Thomson Reuters Eikon fields list specified by the input argument fields
`UpdateMode`	Thomson Reuters Eikon Desktop Data API real-time data update mode

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”
- “Writing and Running Custom Event Handler Functions” on page 1-29

See Also

`getdata` | `history` | `start` | `stop` | `treikon`

start

Start Thomson Reuters Eikon real-time data retrieval

Syntax

```
start(c,subs)
```

Description

`start(c,subs)` starts real-time data retrieval using the Thomson Reuters Eikon connection `c`. `start` starts real-time data retrieval for the securities specified in the Thomson Reuters Eikon real-time subscription list `subs`.

Examples

Start Real-Time Data Retrieval

To start retrieving real-time data, create the connection `c` using `treikon`. Retrieve real-time data using `realtime`. For an example showing these activities, see "Retrieve Thomson Reuters Eikon Real-Time Data".

After stopping real-time data updates, resume real-time data updates using Thomson Reuters Eikon connection `c` and Thomson Reuters Eikon subscription list `subs`.

```
start(c,subs)
```

To close the Thomson Reuters Eikon connection, exit MATLAB.

- "Retrieve Thomson Reuters Eikon Real-Time Data"

Input Arguments

`c` — Thomson Reuters Eikon connection
connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

`subs` — Thomson Reuters Eikon subscription list
structure

Thomson Reuters Eikon subscription list, specified as a structure created using `realtime`.

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”

See Also

`realtime` | `stop` | `treikon`

stop

Stop Thomson Reuters Eikon real-time data retrieval

Syntax

```
stop(c,subs)
```

Description

`stop(c,subs)` stops real-time data retrieval using the Thomson Reuters Eikon connection `c`. `stop` stops real-time data retrieval for the securities specified in the Thomson Reuters Eikon real-time subscription list `subs`.

Examples

Stop Real-Time Data Retrieval

To stop retrieving real-time data, create the connection `c` using `treikon`. Retrieve real-time data using `realtime`. For an example showing these activities, see "Retrieve Thomson Reuters Eikon Real-Time Data".

After retrieving real-time data, stop real-time data updates using Thomson Reuters Eikon connection `c` and Thomson Reuters Eikon subscription list `subs`.

```
stop(c,subs)
```

To close the Thomson Reuters Eikon connection, exit MATLAB.

- "Retrieve Thomson Reuters Eikon Real-Time Data"

Input Arguments

`c` — Thomson Reuters Eikon connection

connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

`subs` — Thomson Reuters Eikon subscription list
structure

Thomson Reuters Eikon subscription list, specified as a structure created using `realtime`.

More About

Tips

- For details about the Thomson Reuters Eikon Desktop Data API, see Thomson Reuters Eikon Help.
- “Workflow for Thomson Reuters Eikon”

See Also

`realtime` | `start` | `treikon`

chain

Retrieve chain data from Thomson Reuters Eikon

Syntax

```
d = chain(c,s)
```

Description

`d = chain(c,s)` retrieves chain data for security `s` using the Thomson Reuters Eikon connection `c`.

Examples

Retrieve Chain Data

Create a Thomson Reuters Eikon connection `c`.

```
c = treikon;
c.DataAPIClass.add_OnStatusChanged(@trestatuseventhandler)
c.DataAPIClass.Status
c.DataAPIClass.Initialize
c.Source = 'IDN';
```

```
ans =

Disconnected

ans =

Succeed

ans =

Connected
```

MATLAB connects to Thomson Reuters Eikon when the Command Window displays this message: Connected.

Retrieve chain data for the UK Pound Sterling/US Dollar FX Spot Rate.

```
s = 'GBP=';

d = chain(c,s)

d = 

    'GBPCONTINFO'
    'GBP='
    'GBP=FXBP'
    ...
```

`chain` returns `d` as a cell array containing the list of instrument names.

To close the Thomson Reuters Eikon connection, exit MATLAB.

- “Retrieve Thomson Reuters Eikon Current Data”

Input Arguments

c — Thomson Reuters Eikon connection

connection object

Thomson Reuters Eikon connection, specified as a connection object created using `treikon`.

s — Security list

string | cell array

Security list, specified as a string for one security or a cell array for multiple securities.

Data Types: `char` | `cell`

Output Arguments

d — Return data

cell array

Return data, returned as a cell array containing a list of instrument names.

More About

- “Workflow for Thomson Reuters Eikon”

rnseloader

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

`rdthloader` | `reuters`

tlkrs

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

close

Close connection to SIX Financial Information

Syntax

```
close(C)
```

Description

`close(C)` closes the connection, `C`, to SIX Financial Information.

See Also

`tlkrs`

getdata

Current SIX Financial Information data

Syntax

```
D = getdata(c,s,f)
```

Description

`D = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`.

Examples

Retrieve SIX Financial Information pricing data for specified securities.

```
% Connect to Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert specified fields to ID strings.
ids =  tkfieldtoid(c,{'Bid','Ask','Last'},'market');

% Retrieve data for specified securities.
d = getdata(c,{'1758999,149,134','275027,148,184'},ids);
```

Your output appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, and `d.P` contains the pricing data.

View the instrument IDs like this:

```
d.I.k
```

```
ans =
    '1758999,149,134'
    '275027,148,184'
```

View the pricing data field IDs like this:

```
d.P.k

ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
    '33,3,1'
    '3,1,1'
```

And the pricing data like this:

```
d.P.v

ans =

    '44.94'
    '44.95'
    []
    '0.9715'
    '0.9717'
    []
```

Convert field IDs in `d.P.k` to field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'market')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names (`Bid`, `Ask`, `Last`) and corresponding IDs.

See Also

`tlkrs` | `timeseries` | `tkidtofield` | `history` | `tkfieldtoid`

history

End of day SIX Financial Information data

Syntax

```
D = history(c,s,f,fromdate,todate)
```

Description

`D = history(c,s,f,fromdate,todate)` returns the historical data for the security list `s`, for the fields `f`, for the dates `fromdate` to `todate`.

Examples

Retrieve end of day SIX Financial Information data for the specified security for the past 5 days.

```
c = tlkrs('US12345','userapid01','userapid10')
ids = tkfieldtoid(c,{'Bid','Ask'},'history');
d = history(c,{'1758999,149,134'},ids,floor(now)-5,floor(now));
```

```
d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
     HL: [1x1 struct]
     HD: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.HD` contains the dates, and `d.P` contains the pricing data.

View the dates:

```
d.HD.d
```

```
ans =
    '20110225'
    '20110228'
    '20110301'
```

View the pricing field IDs:

```
d.P.k

ans =

    '3,2'
    '3,3'
    '3,2'
    '3,3'
    '3,2'
    '3,3'
```

View the pricing data:

```
d.P.v

ans =

    '45.32'
    '45.33'
    '45.26'
    '45.27'
    '44.94'
    '44.95'
```

Convert the field identification strings in `d.P.k` to their corresponding field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

See Also

`tlkrs` | `timeseries` | `tkidtofield` | `getdata` | `tkfieldtoid`

isconnection

Determine if SIX Financial Information connection is valid

Syntax

```
X = isconnection(C)
```

Description

`X = isconnection(C)` returns `true` if `C` is a valid SIX Financial Information connection and `false` otherwise.

See Also

`tlkrs` | `close` | `getdata`

timeseries

SIX Financial Information intraday tick data

Syntax

```
D = timeseries(c,s,t)
D = timeseries(c,s,{startdate,enddate})
D = timeseries(c,s,t,5)
```

Description

`D = timeseries(c,s,t)` returns the raw tick data for the SIX Financial Information connection object `c`, the security `s`, and the date `t`. Every trade, best, and ask tick is returned for the given date or date range.

`D = timeseries(c,s,{startdate,enddate})` returns the raw tick data for the security `s`, for the date range defined by `startdate` and `enddate`.

`D = timeseries(c,s,t,5)` returns the tick data for the security `s`, for the date `t` in intervals of 5 minutes, for the field `f`. Intraday tick data requested is returned in 5-minute intervals, with the columns representing:

- First
- High
- Low
- Last
- Volume weighted average
- Moving average

Examples

Retrieve SIX Financial Information intraday tick data for the past 2 days:

```
c = tlkrs('US12345','userapid01','userapid10')
```

```
d = timeseries(c,{'1758999,149,134'}, ...
   {floor(now)-.25,floor(now)})
```

Display the returned data:

```
d = 

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
    TSL: [1x1 struct]
     TS: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.TS` contains the date and time data, and `d.P` contains the pricing data.

Display the tick times:

```
d.TS.t(1:10)

ans = 

    '013500'
    '013505'
    '013510'
    '013520'
    '013530'
    '013540'
    '013550'
    '013600'
    '013610'
    '013620'
```

Display the field IDs:

```
d.P.k(1:10)

ans = 

    '3,4'
    '3,2'
    '3,3'
    '3,4'
    '3,2'
    '3,3'
    '3,4'
    '3,2'
    '3,3'
    '3,4'
```

Convert these IDs to field names (`Mid`, `Bid`, `Ask`) with `tkidtofield`:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names and corresponding IDs.

Display the corresponding tick values:

```
d.P.v(1:10)

ans =

    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
```

See Also

`tlkrs` | `history` | `getdata`

tkfieldtoid

SIX Financial Information field names to identification string

Syntax

```
D = tkfieldtoid(c,f,typ)
```

Description

`D = tkfieldtoid(c,f,typ)` converts SIX Financial Information field names to their corresponding identification strings. `c` is the SIX Financial Information connection object, `f` is the field list, and `typ` denotes the field. Options for the field include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with `history`.

Examples

Retrieve pricing data associated with specified identification strings:

```
% Connect to SIX Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert field names to identification strings.
ids = tkfieldtoid(c,{'bid','ask','last'},'market');

% Retrieve data associated with the identification strings.
d = getdata(c,{'1758999,149,134','275027,148,184',ids);
```

See Also

`tlkrs` | `history` | `tkidtofield` | `getdata` | `timeseries`

tkidtofield

SIX Financial Information identification string to field name

Syntax

```
D = tkidtofield(c,f,typ)
```

Description

`D = tkidtofield(c,f,typ)` converts SIX Financial Information field identification strings to their corresponding field names. `c` is the SIX Financial Information connection object, `f` is the ID list, and `typ` denotes the fields. Options for the fields include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with the `history`.

Examples

When you retrieve output from SIX Financial Information, it appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

The instrument IDs are found in `d.I`, and the pricing data is found in `d.P`. The output for `d.P.k` appears like this:

```
ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
    '33,3,1'
    '3,1,1'
```

Convert the field IDs in `d.P.k` to their field names with `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` | `history` | `tkfieldtoid` | `getdata` | `timeseries`

yahoo

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.

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)
```

Output Arguments

c — Yahoo! connection

connection object

Yahoo! connection, returned as a connection object.

See Also

`builduniverse` | `close` | `fetch` | `get` | `isconnection` | `trpdata`

builduniverse

Retrieve total return price data from Yahoo!

Syntax

```
data = builduniverse(c,s,fromdate,todate,period)
```

Description

`data = builduniverse(c,s,fromdate,todate,period)` retrieves total return price series data for security `s` using the Yahoo! connection `c`. Retrieve data starting from the date `fromdate` through `todate` using the periodicity `period` to denote the data frequency.

Examples

Compute a Total Return Price Series

Connect to Yahoo! Finance.

```
c = yahoo;
```

Create a security list for Google.

```
s = {'GOOG'};
```

Retrieve a daily total return price series for Google starting January 15, 2012 through today. The total is calculated from prices, splits, and dividends.

```
fromdate = '1/15/2012';
todate = floor(now);

data = builduniverse(c,s,fromdate,todate);
```

Display the data.

```
data
```

```
data =
    734885.00          1.00
    734886.00          1.01
    734887.00          1.02
    ...
```

`data` contains the numeric representation of the date in the first column and the total return prices for Google in the second column.

Close the connection.

```
close(c)
```

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 multiple securities. Security strings must be in a format recognizable by the Yahoo! server.

Data Types: `char` | `cell`

`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
`'d'` (default) | `'w'` | `'m'`

Period within a date range, specified as one of these enumerated strings. Values are:

Enumerated String	Description
`'d'`	Daily
`'w'`	Weekly
`'m'`	Monthly

Data Types: `char`

Output Arguments

`data` — Total return price series
matrix

Total return price series, returned as an m-by-(n + 1) matrix, where m refers to the number of records of data and n refers to the number of securities. The first column of the matrix contains MATLAB date numbers and the remaining columns are the total return prices for each security.

More About

Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by the `builduniverse` function contains the expected results.

See Also

`fetch` | `trpdata`

close

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`

fetch

Request data from Yahoo! Finance

Syntax

```
d = fetch(c,s)
d = fetch(c,s,date)
d = fetch(c,s,fromdate,todate)
d = fetch(c,s,fromdate,todate,period)

d = fetch(c,s,f)
d = fetch(c,s,f,date)
d = fetch(c,s,f,fromdate,todate)
d = fetch(c,s,f,fromdate,todate,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(c,s,fromdate,todate,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(c,s,f,fromdate,todate,period)` returns security data with the indicated period.

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     7079500.00          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     7673400.00            94.80
    730533.00        116.50        118.87        115.75        116.75     5237500.00            98.38
    730532.00        116.50        117.31        115.25        115.75     3966900.00            97.54
    730531.00        115.87        117.44        113.87        117.12     5177500.00            98.69
    730530.00        116.00        116.37        114.50        116.06     4544000.00            97.80
    ...
```

`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)
```

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.

See Also

`close` | `get` | `isconnection` | `yahoo`

get

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

Determine if 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`

trpdata

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.

More About

Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by the `trpdata` function contains the expected results.

See Also

`close` | `yahoo`