Datafeed Toolbox™

User's Guide

MATLAB®

R2017a

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)
September 2015	Online only	Revised for Version 5.2 (Release 2015b)
March 2016	Online only	Revised for Version 5.3 (Release 2016a)
September 2016	Online only	Revised for Version 5.4 (Release 2016b)
March 2017	Online only	Revised for Version 5.5 (Release 2017a)

Contents

1 Getting Started

2 Communicate with Financial Data Servers

3 Data Provider Workflows

4 Troubleshooting

5 Money.Net Topics

6 Thomson Reuters Elektron Topics

7 Datafeed Toolbox Graphical User Interface

8 Functions — Alphabetical List

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®, 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, RMDS, Datastream®, NewsScope, and Tick History connectivity
- Connectivity to FactSet, 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 "Communicating 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.
- FactSet
  - A license to use FactSet DataDirect or FactSet Workstation.
- Haver Analytics
- IQFEED
- Kx Systems, Inc.
- RavenPack® News Analytics
  - For details about setup and configuration, see the RavenPack Developer Zone.
- 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.

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
- STATS.com
- Thomson Reuters Datastream
- Thomson Reuters Tick History
- Yahoo!

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

Platform Requirements

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

- Bloomberg
- Haver Analytics
- IQFEED

More About

- “Communicating with Data Providers” on page 2-2
- “Installing Bloomberg and Configuring Connections” on page 1-5
- “Configuring Reuters Market Data System Connections” on page 1-9

Installing Bloomberg and Configuring Connections

In this section…

Datafeed Toolbox provides various ways to connect to Bloomberg. For details, see "Comparing Bloomberg Connections" on page 2-4. Before connecting to Bloomberg, follow these required steps.

Install Bloomberg Software

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

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

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

Add JAR Files to the MATLAB Java Class Path

Bloomberg Desktop, Bloomberg Server, and Bloomberg B-PIPE

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

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

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

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

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

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

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

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

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

Bloomberg Data License

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

Troubleshooting Adding JAR Files

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

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

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

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

Run Bloomberg Communications Server

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

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

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

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

See Also

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

Related Examples

- “Connect to Bloomberg” on page 3-2

More About

- “Data Server Connection Requirements” on page 1-3
- “Communicating with Data Providers” on page 2-2
- “Workflow for Bloomberg” on page 3-20

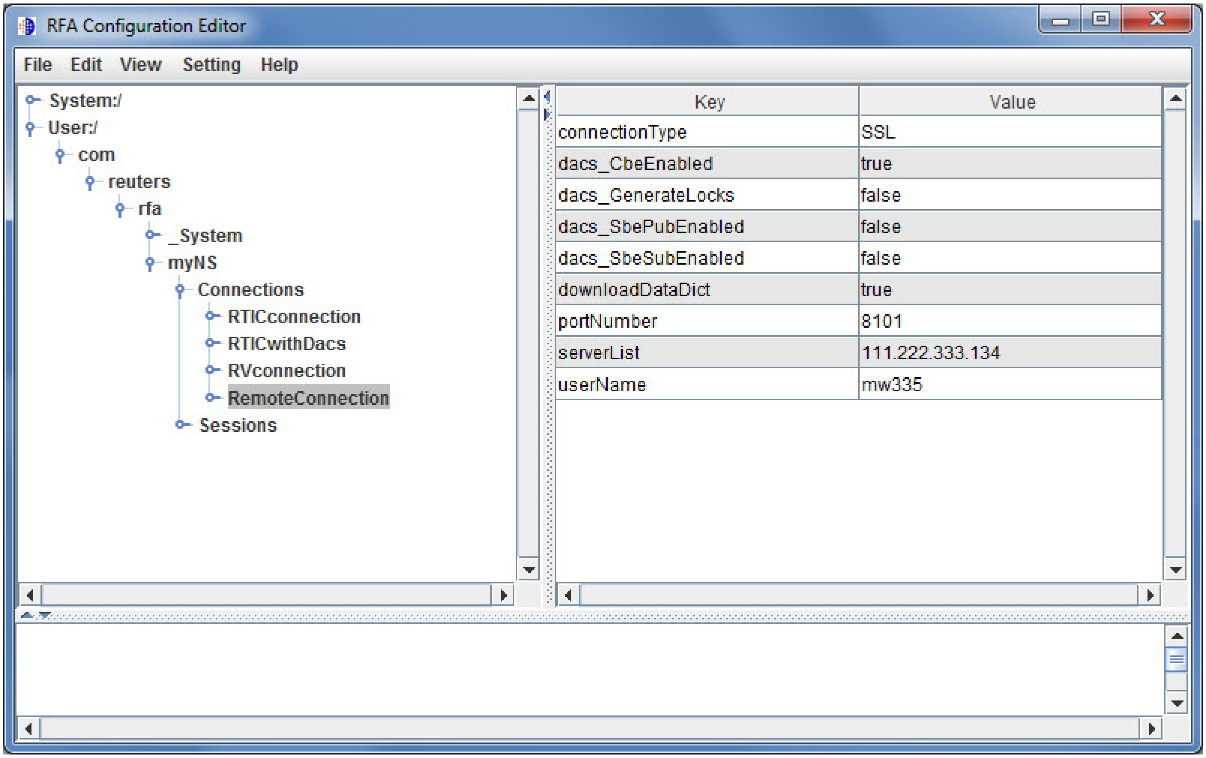
Configuring Reuters Market Data System Connections

In this section...

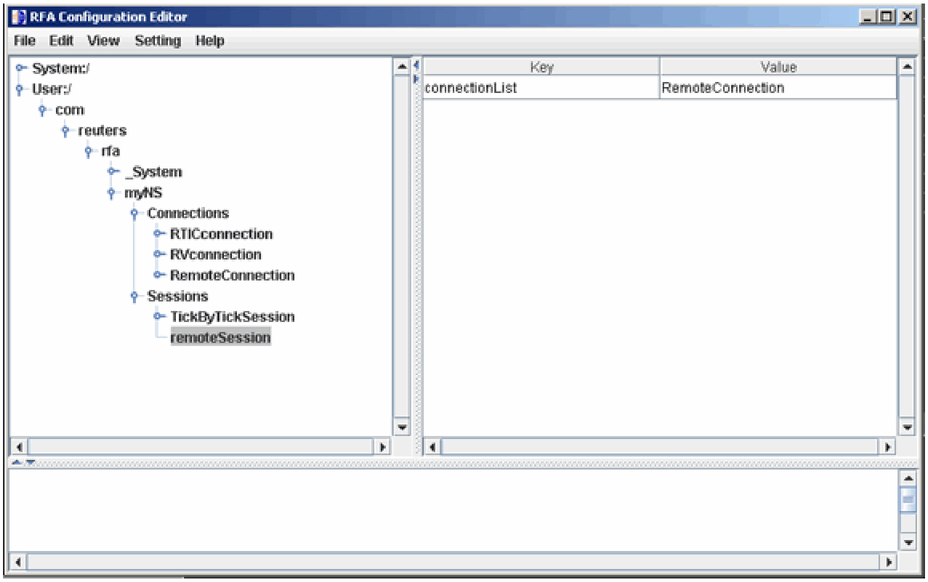
To connect to the Reuters® Market Data System for the first time or change the authentication type, configure the Reuters connection using the RFA Configuration Editor.

Configure a Reuters Connection

1. Open the 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 obtained from Reuters.
4. Define a namespace, connection, and session associated with the connection `RemoteConnection`. Set the key and value fields as shown in the RFA Configuration Editor.



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



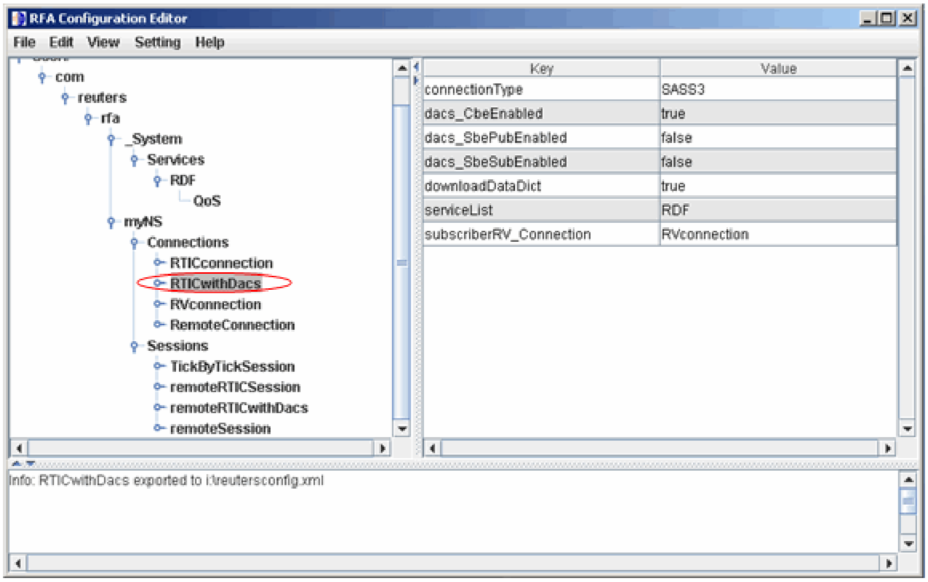
5 To connect without Data Access Control System (DACS) authentication, disable DACS by setting the keys in `RemoteConnection` to the values as shown in this table.

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

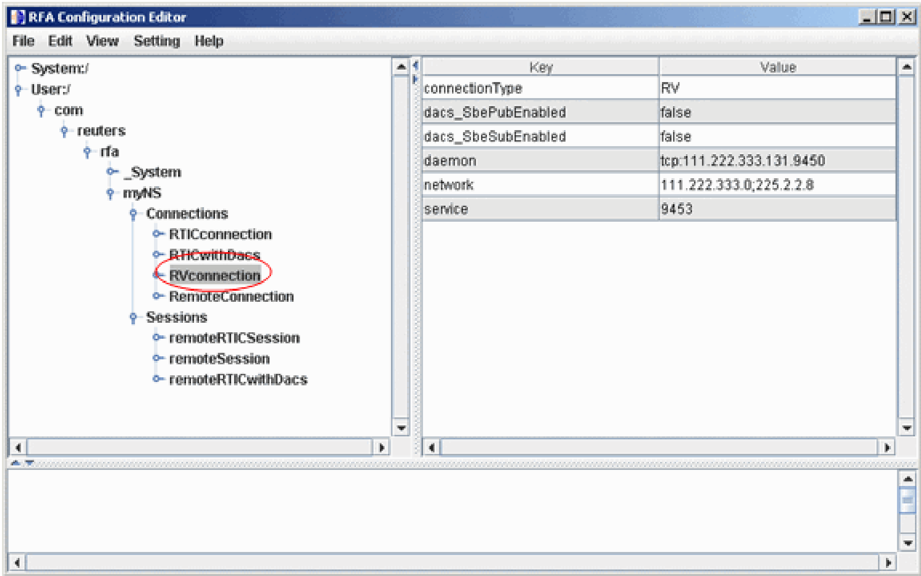
6 To run an SSL connection, set the key `dacs_GenerateLocks` to the value `false` in `RemoteConnection`.

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

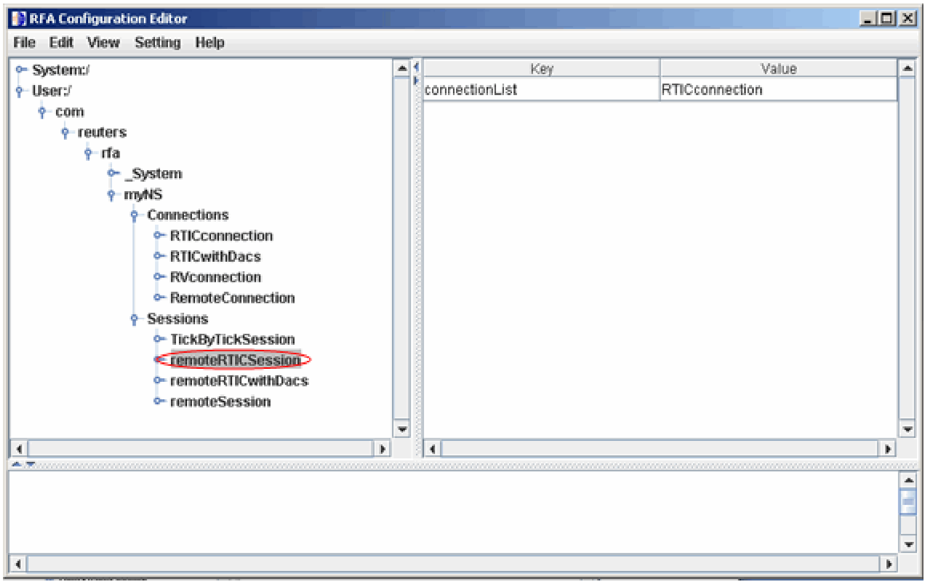
1 After loading and modifying the sample configuration file, set the keys and value fields as shown in the RFA Configuration Editor for the connection `RTICwithDacs`.



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

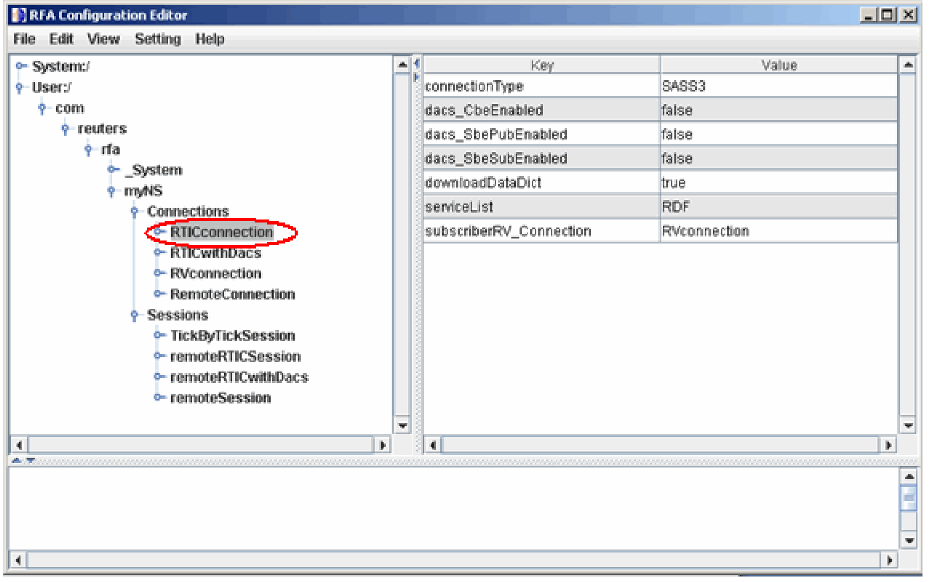


The RFA Configuration Editor shows the session `remoteRTICSession` referencing the `RTICConnection`.

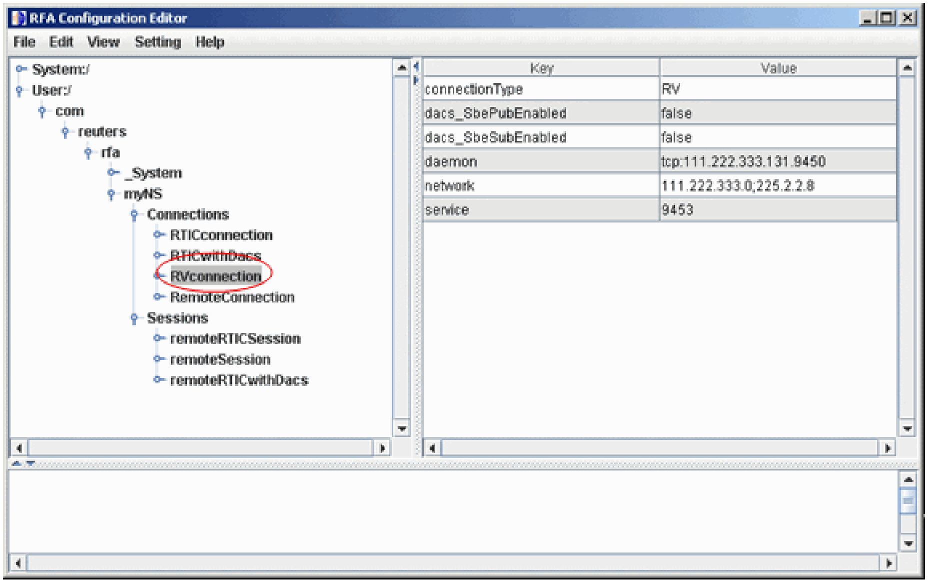


Configure an RTIC (TIC-RMDS Edition) Reuters Connection Without DACS Authentication

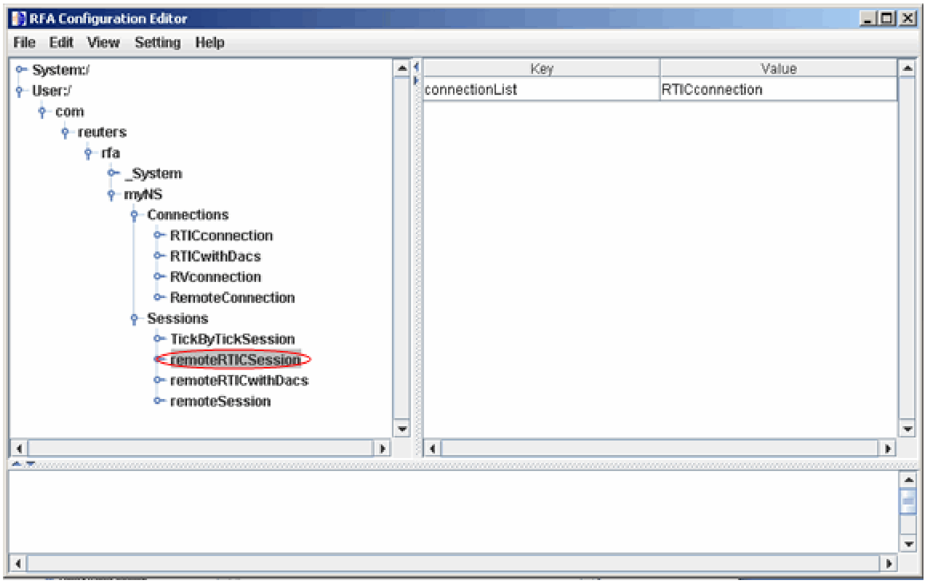
1 After loading and modifying the sample configuration file, set the key and value fields as shown for `RTICConnection`.



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



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



Troubleshoot the Reuters Configuration Editor

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

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

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

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

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

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

  ```
  SEVERE: com.reuters.rfa.entitlements._Default.Global
  DACS initialization failed:
  com.reuters.rfa.dacs.AuthorizationException:
  Cannot start the DACS Library thread due to -
  Cannot locate JNI library - RFADacsLib
  ```
  then add an entry to the $MATLAB/toolbox/local/librarypath.txt file that points to the folder containing these files:

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

See Also

reuters

More About

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

Retrieve Current and Historical Data Using Bloomberg

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

Connect to Bloomberg®

Create a Bloomberg® Desktop connection.

```
c = blp;
```

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

Retrieve Current Data

Format MATLAB® data display for currency.

```
format bank
```

Retrieve closing and open prices for Microsoft®.

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

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

```
d = 

  struct with fields:

    LAST_PRICE: 62.32
          OPEN: 62.48


sec = 

  cell
```

```
    'MSFT US Equity'
```

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

Retrieve Historical Data

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

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

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


d =

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


sec =

  cell

    'MSFT US Equity'
```

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

Find Maximum Open Price in Date Range

Calculate the maximum open price for the year 2012.

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

```
ans =

        32.22
```

Close Bloomberg® Connection

```
close(c)
```

See Also

blp | close | getdata | history

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Bloomberg Historical Data” on page 3-10
- “Retrieve Bloomberg Intraday Tick Data” on page 3-13
- “Retrieve Bloomberg Real-Time Data” on page 3-15

Retrieve 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. Before connecting to Thomson Reuters, configure the RMDS connections. For details, see "Configuring Reuters Market Data System Connections" on page 1-9.

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`

Retrieve Historical Data Using FRED®

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

Connect to the FRED® data server using the URL `'https://research.stlouisfed.org/fred2/'`.

```
url = 'https://research.stlouisfed.org/fred2/';
c = fred(url);
```

Adjust the display data format for currency.

```
format bank
```

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

```
series = 'DEXUSEU';

d = fetch(c,series)
```

```
d = 

  struct with fields:

                 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 2016-07-08'
           LastUpdated: ' 2016-07-11 3:41 PM CDT'
                 Notes: ' Noon buying rates in New York City for cable tra...'
                  Data: [4570×2 double]
```

`d` contains the series description.

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

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

```
ans =

     730124.00          1.18
     730125.00          1.18
     730126.00          1.16
```

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

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

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

```
d = 

  struct with fields:

                  Title: ' U.S. / Euro Foreign Exchange Rate'
               SeriesID: ' DEXUSEU'
                 Source: ' Board of Governors of the Federal Reserve System...'
                Release: ' H.10 Foreign Exchange Rates'
     SeasonalAdjustment: ' Not Seasonally Adjusted'
              Frequency: ' Daily'
                  Units: ' U.S. Dollars to One Euro'
              DateRange: ' 1999-01-04 to 2016-07-08'
            LastUpdated: ' 2016-07-11 3:41 PM CDT'
                  Notes: ' Noon buying rates in New York City for cable tra...'
                   Data: [110×2 double]
```

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

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

```
maxforex =

          1.35
```

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

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

```
ans =

24-Feb-2012
```

Close the FRED® connection.

```
close(c)
```

See Also

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

Retrieve Historical Data Using Haver Analytics

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

Connect to Haver Analytics

Connect to Haver Analytics using a daily file.

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

Retrieve All Historical Data

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

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

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

Display the first three rows of data.

```
d(1:3,:)

ans =

     715511.00           2.38
     715512.00           2.50
     715515.00           2.50
```

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

Retrieve Historical Data for a Date Range

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

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

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

Close the Haver Analytics Connection

```
close(c)
```

Open the Haver Analytics User Interface

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

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

```
havertool(c)
```

For details, see the `havertool` function.

See Also

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

Retrieve Intraday and Historical Data Using IQFEED

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

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`

Retrieve 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 currently available data for IBM.

```
sec = 'IBM';
d = fetch(c,sec)
```

```
d = 

    Symbol: {'IBM'}
      Last: 163.1720
      Date: 736047
      Time: 0.4333
    Change: -1.4580
      Open: 164.7900
      High: 164.8900
       Low: 163.1500
    Volume: 658816
```

`d` is a structure containing the current data with these fields. Access the fields of `d` using dot notation: `d.Low`.

- Stock symbol.
- Price at the last tick.
- `datetime` value of the data-retrieval date. To convert `d.Date` to a character vector, use `datestr`.
- `datetime` value of the data-retrieval time.
- Dollar value change in stock price from the closing price of the previous day.
- Opening price.
- Highest price yet for the current day.
- Lowest price yet for the current day.
- Current volume.

When you retrieve the data for the current moment, the results vary because they depend on the retrieval date and time.

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` | `datestr` | `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" (MATLAB). 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 character vector. For other functions, specify the custom event handler function name either as a character vector or function handle. For details about function handles, see "Create Function Handle" (MATLAB).

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

Use a character vector.

```
[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" (MATLAB)
- "Create Function Handle" (MATLAB)

Communicate with Financial Data Servers

Communicating with Data Providers

Datafeed Toolbox supports connection to these data providers. This table lists the connection functions for each data provider.

Data Provider	Website	Connection Function	API Documentation
Bloomberg	http://www.bloomberg.com	`blp`, `blpsrv`, `bpipe`, or `bdl`	*Bloomberg API Developer's Guide* using the **WAPI <GO>** option from the Bloomberg terminal
FactSet	http://www.factset.com	`factset` or `fds`	Not available
FRED	https://research.stlouisfed.org/fred2/	`fred`	Not available
Haver Analytics	http://www.haver.com	`haver`	Not available
IQFEED	http://www.iqfeed.net/	`iqf`	IQFEED API
Kx Systems, Inc.	http://www.kx.com	`kx`	Kx Systems, Inc., GitHub Repository
Money.Net	https://www.money.net	`moneynet`	Money.Net API Documentation
RavenPack News Analytics	http://www.ravenpack.com/products/ravenpack-news-analytics/	`ravenpack`	RavenPack Developer Zone Overview
SIX Financial Information	http://www.six-financial-information.com	`tlkrs`	Not available
STATS.com	http://www.stats.com	`statsllc`	Not available
Thomson Reuters	http://www.thomsonreuters.	`datastream`, `reuters`, or `rdth`	Not available

Data Provider	Website	Connection Function	API Documentation
Yahoo!	http://finance.yahoo.com	`yahoo`	Not available

More About

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

Comparing Bloomberg Connections

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

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

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

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

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

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

Related Examples

- "Connect to Bloomberg" on page 3-2

More About

- "Installing Bloomberg and Configuring Connections" on page 1-5

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

3

Data Provider Workflows

Connect to Bloomberg

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

Create the Bloomberg Desktop Connection

```
c = blp
```

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

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

Validate the connection `c`.

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

```
v =

     1
```

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

Retrieve the Bloomberg Desktop connection properties.

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

```
v =

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

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

Close the Bloomberg Desktop connection.

```
close(c)
```

Create the Bloomberg Server Connection

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

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

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

c = blpsrv(uuid,serverip)

c = 

  blpsrv with properties:

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

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

Close the Bloomberg Server connection.

```
close(c)
```

Create the B-PIPE Connection

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

- The authentication is Windows Authentication by setting `authorizationtype` to `'OS_LOGON'`.

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

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

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

```
c = 

  bpipe with properties:

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

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

Close the B-PIPE connection.

```
close(c)
```

Create the Bloomberg Data License Connection

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

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

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

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

c =

bdl with properties:

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

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

Close the Bloomberg Data License connection.

```
close(c)
```

See Also

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

Related Examples

- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Bloomberg Historical Data” on page 3-10
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19
- “Retrieve Bloomberg Intraday Tick Data” on page 3-13
- “Retrieve Bloomberg Real-Time Data” on page 3-15
- “Retrieve Data Using Bloomberg Data License” on page 3-17

More About

- “Data Server Connection Requirements” on page 1-3

- “Comparing Bloomberg Connections” on page 2-4
- “Workflow for Bloomberg” on page 3-20

Retrieve Bloomberg Current Data

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

Connect to Bloomberg

Create a Bloomberg Desktop connection.

```
c = blp;
```

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

Retrieve Current Data for Single Security

Retrieve last and open prices for Microsoft®.

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

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

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

```
d = 

  struct with fields:

    LAST_PRICE: 62.30
          OPEN: 62.95


sec = 

  cell

    'MSFT US Equity'
```

Retrieve Current Data for Multiple Securities

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

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

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

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

d = 

  struct with fields:

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


sec =

  2×1 cell array

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

Display the last price for both securities.

```
d.LAST_PRICE

ans =

        166.73
         12.63
```

Close Bloomberg Connection

```
close(c)
```

See Also

blp | close | getdata

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Historical Data” on page 3-10

- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19
- "Retrieve Bloomberg Intraday Tick Data" on page 3-13
- "Retrieve Bloomberg Real-Time Data" on page 3-15

More About

- "Workflow for Bloomberg" on page 3-20

Retrieve Bloomberg Historical Data

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

Connect to Bloomberg

Create a Bloomberg Desktop connection.

```
c = blp;
```

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

Retrieve Historical Data for One Security

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

`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.

```
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 =

  cell

    'MSFT US Equity'
```

Retrieve Historical Data for Multiple Securities

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

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

```
s = {'IBM US Equity','F US Equity'};
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,s,fields,fromdate,todate,period)

d =

  2×1 cell array

    [12×3 double]
    [12×3 double]


sec =

  2×1 cell array

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

Display the closing and open prices for the first security.

The data in the double matrix is:

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

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

```
d{1}
```

```
ans =

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

Close Bloomberg Connection

```
close(c)
```

See Also

`blp` | `close` | `history`

Related Examples

- "Connect to Bloomberg" on page 3-2
- "Retrieve Bloomberg Current Data" on page 3-7
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19
- "Retrieve Bloomberg Intraday Tick Data" on page 3-13
- "Retrieve Bloomberg Real-Time Data" on page 3-15

More About

- "Workflow for Bloomberg" on page 3-20

Retrieve Bloomberg Intraday Tick Data

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

```
ans =

  Columns 1 through 7

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

Column 8

   38902968.00
    8779374.00
    9626896.00
   ...
```

The columns in `d` contain the following:

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

The first row of data shows 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 3-7
- "Retrieve Bloomberg Historical Data" on page 3-10
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19
- "Retrieve Bloomberg Real-Time Data" on page 3-15

More About

- "Workflow for Bloomberg" on page 3-20

Retrieve Bloomberg Real-Time Data

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

`v3stockticker` requires the input argument `f` of `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 3-7
- "Retrieve Bloomberg Historical Data" on page 3-10
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19
- "Retrieve Bloomberg Intraday Tick Data" on page 3-13

More About

- "Workflow for Bloomberg" on page 3-20
- "Writing and Running Custom Event Handler Functions" on page 1-35

Retrieve Data Using Bloomberg Data License

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

Connect to Bloomberg Data License

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

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

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

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

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

Request Bloomberg Data File

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

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

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

```
s = dir(c)

s =
```

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

The output file `getdataoutput.out` is available.

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

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

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

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

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

Import Bloomberg Data into MATLAB

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

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

```
d = 

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

`d` is a structure with these fields:

- Output file header information
- Security identifier
- Return code

- Number of fields requested and received
- Open price
- Last price
- High price
- Low price
- Date of last close

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

Display the output file header information.

```
d.Header
```

```
ans = 

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

Close Bloomberg Data License Connection

```
close(c)
```

See Also

`bdl` | `close` | `dir`

Related Examples

- “Connect to Bloomberg” on page 3-2

More About

- “Workflow for Bloomberg” on page 3-20

Workflow for Bloomberg

In this section...
"Bloomberg Desktop, Bloomberg Server, or Bloomberg B-PIPE Services" on page 3-20
"Bloomberg Data License Service" on page 3-21

You can use Bloomberg to monitor market price information.

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

Connect to Bloomberg

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

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

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

Close Bloomberg Connection

Close the Bloomberg connection by using `close`.

Bloomberg Data License Service

To connect and retrieve data using Bloomberg Data License:

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

Related Examples

- “Connect to Bloomberg” on page 3-2
- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Bloomberg Historical Data” on page 3-10
- “Retrieve Bloomberg Intraday Tick Data” on page 3-13
- “Retrieve Bloomberg Real-Time Data” on page 3-15
- “Retrieve Data Using Bloomberg Data License” on page 3-17

More About

- “Comparing Bloomberg Connections” on page 2-4

Determine the Event Volume Indicator Using RavenPack News Analytics

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

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

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

The variables are:

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

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

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

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

Connect to RavenPack News Analytics

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

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

Retrieve Historical and Intraday RavenPack News Data

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

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

e is a table that contains the entitlement data.

Retrieve the RavenPack News Analytics symbol for equities data.

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

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

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

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

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

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

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

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

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

Retrieve Twitter News Data

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

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

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

```
iIT = strcmp('insider-trading',companyData.GROUP);
```

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

Determine EVI for Twitter News Data

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

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

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

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

Clear temporary MATLAB variables.

```
clear essTimestamp EVI eviData
```

Retrieve Real-Time Twitter News Data

Populate real-time Twitter EVI data into the MATLAB variable `twitter` in the Workspace browser using `realtime`. Run `realtime` using the RavenPack News

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

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

Close RavenPack News Analytics Connection

```
close(c)
```

References

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

See Also

close | entitlements | ravenpack | realtime | rploader | timeseries

More About

- “Workflow for RavenPack News Analytics” on page 3-26

External Websites

- RavenPack Developer Zone Overview

Workflow for RavenPack News Analytics

You can use RavenPack News Analytics to retrieve news data.

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

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

Related Examples

- “Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-22

Compare Player Salaries by Injury Status

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

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

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

Connect to STATS.com

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

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

Retrieve Baseball Player Salary and Injury Data

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

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

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

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

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

`salaryData` contains these column names:

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

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

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

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

data = fetch(sBaseball);
```

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

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

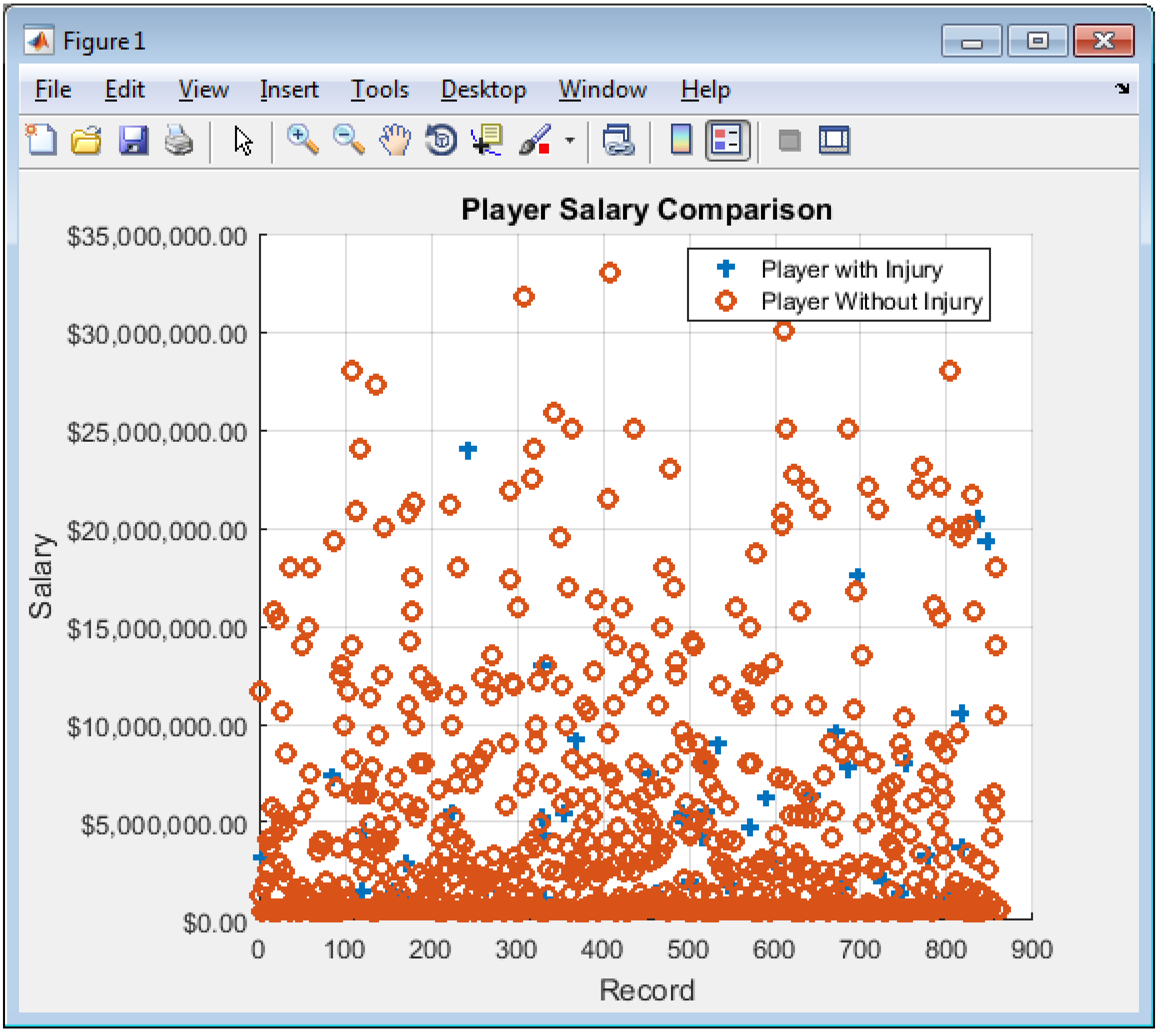
Visualize Baseball Player Salary and Injury Data

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

```
sInjured = scatter(salaryData.Record(salaryData.Injured), ...
    salaryData.Salary(salaryData.Injured),'Marker','+', ...
    'Linewidth',2);
hold on
sNotInjured = scatter(salaryData.Record(~salaryData.Injured), ...
    salaryData.Salary(~salaryData.Injured),'Marker','O', ...
    'Linewidth',2);
hold off
sInjured.MarkerFaceColor = 'flat'
```

```
grid on
b = gca
title('Player Salary Comparison')
xlabel('Record')
ylabel('Salary')
ytickformat(b,'usd')
legend({'Player with Injury','Player Without Injury'},'Location','best')
```

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



See Also

`fetch` | `statsllc`

More About

- "Retrieve Team Standings for the Current Year" on page 3-32

- "STATS.com Data Retrieval Errors" on page 4-2

External Websites

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

Retrieve Team Standings for the Current Year

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

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

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

Connect to STATS.com

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

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

Retrieve Team Standings Data

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

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

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

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

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

    % Iterate through each division
    for j = 1:numDivisions

        % Access team data in each division
        t = struct2table(data.apiResults.league.season.eventType.conferences(i).divisio
        numTeams = length(data.apiResults.league.season.eventType.conferences(i).divis
```

```
        recordTable = [];

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

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

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

```
Eastern Conference Atlantic Division

            Team            Wins    Losses    Percentage    GamesBehind
    ____________________    ____    ______    __________    ___________

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

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

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

- `GamesBehind` — Number of games behind

See Also

`fetch` | `statsllc`

More About

- "Compare Player Salaries by Injury Status" on page 3-27
- "STATS.com Data Retrieval Errors" on page 4-2

External Websites

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

Troubleshooting

STATS.com Data Retrieval Errors

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

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

STATS.com Web Service Error	Probable Causes	Resolution
		• Verify the query parameters. Check which query parameters are included in your STATS.com license.

Related Examples

- “Compare Player Salaries by Injury Status” on page 3-27
- “Retrieve Team Standings for the Current Year” on page 3-32

External Websites

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

Retrieve Thomson Reuters Datastream Historical Data

This example shows how to retrieve historical data from Thomson Reuters Datastream for different fields, date ranges, periods, and currency.

The example uses different Thomson Reuters Datastream codes for securities and field values for data fields. To find codes for other securities and values for other data fields, contact Thomson Reuters.

Connect to Thomson Reuters Datastream

Create a Thomson Reuters Datastream connection using a user name, password, source, and Thomson Reuters URL. `c` is the `datastream` connection object.

```
username = 'DS:username';
password = 'password';
source = 'Datastream';
url = 'http://dataworks.thomson.com/Dataworks/Enterprise/1.0';

c = datastream(username,password,source,url);
```

Retrieve Historical Yearly Data

Retrieve closing price data for the IBM security in the last year using the Thomson Reuters Datastream connection.

`d` is a structure that contains these fields:

- `Source` — Thomson Reuters Datastream source
- `Instrument` — Thomson Reuters Datastream query
- `StatusType` — Thomson Reuters Datastream connection status type
- `StatusCode` — Thomson Reuters Datastream connection status code
- `StatusMessage` — Thomson Reuters Datastream connection status message
- `CCY` — Currency
- `DATE` — Date timestamp for each trading day
- `DISPNAME` — Thomson Reuters security display name
- `FREQUENCY` — Retrieved data frequency (e.g., `'D'` is daily)
- `P` — Closing price data

- `SYMBOL` — Thomson Reuters security symbol name

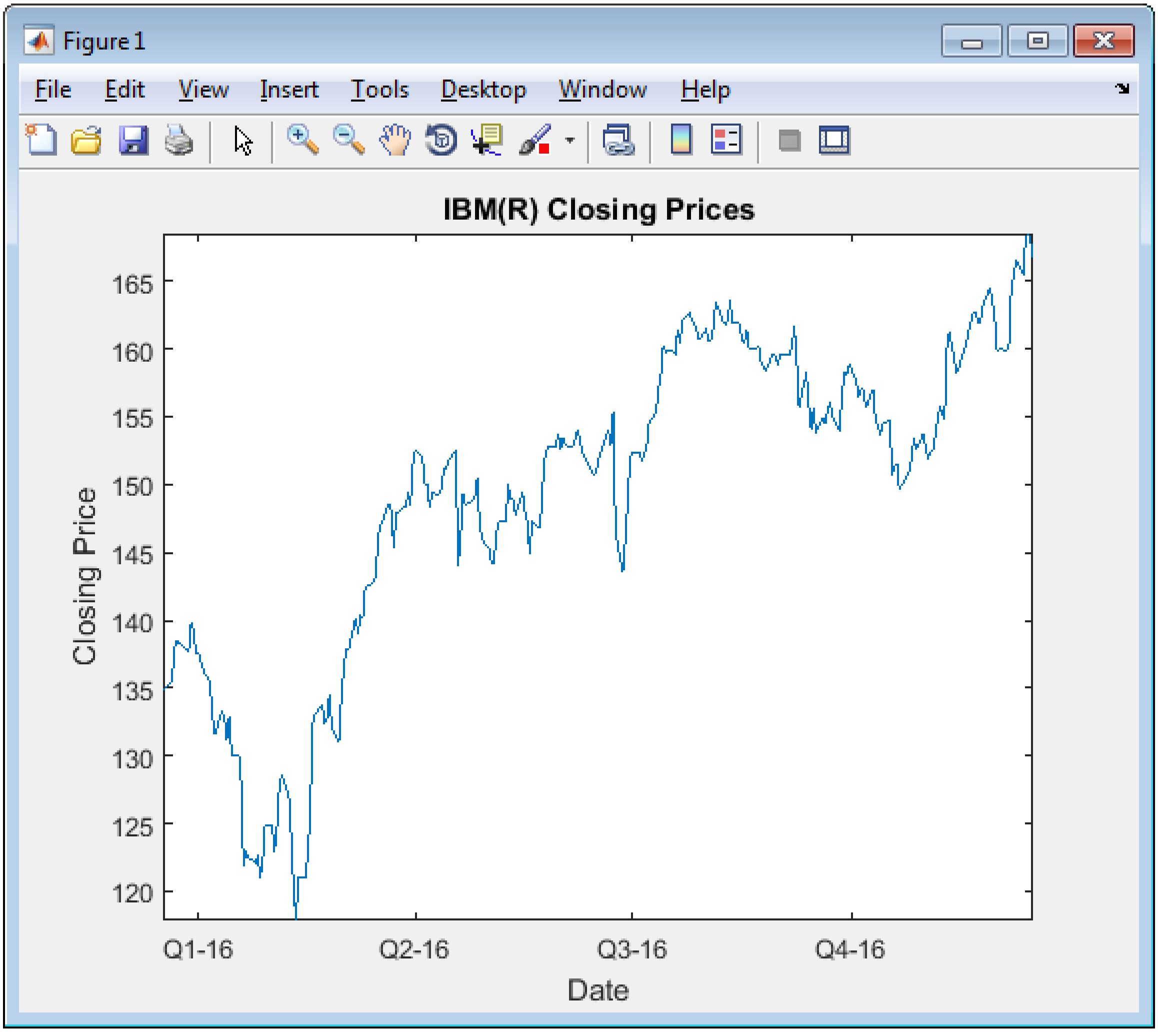
```
sec = 'U:IBM';

d = fetch(c,sec)

d =

  struct with fields:

          Source: 'Datastream'
      Instrument: 'U:IBM'
      StatusType: 'Connected'
      StatusCode: '0'
   StatusMessage: ''
             CCY: 'U$'
            DATE: {261×1 cell}
        DISPNAME: 'INTERNATIONAL BUS.MCHS.'
       FREQUENCY: 'D'
               P: {261×1 cell}
          SYMBOL: 'U:IBM'
```

Thomson Reuters Datastream returns numeric data as character vectors. Convert character vector data to doubles and date timestamps using the `str2num` function. Use the `cellfun` function to convert each character vector in the data. Replace all occurrences of `T` in the date timestamp data to a blank space.

```
IBM = cellfun(@str2num,d.P);
dt = strrep(d.DATE,'T',' ');
date = datenum(dt);
```

Create a plot of the daily closing prices for the IBM security in the last year.

```
plot(date,IBM)
ylabel('Closing Price')
xlabel('Date')
title('IBM(R) Closing Prices')
datetick('x','QQ-YY')
axis tight
```



Retrieve Historical Data for Multiple Fields

Retrieve closing and lowest prices for yesterday. Each price value returns as a field in the structure d.

```
prices = {'P','PL'};
yesterday = floor(now)-1;
```

```
d = fetch(c,sec,prices,yesterday);
```

Convert closing and lowest prices from character vectors to doubles and display them using the `str2num` function.

```
close = str2num(d.P);
low = str2num(d.PL);

[close low]

ans =

        166.73        166.06
```

Retrieve Historical Data for Date Range

Retrieve daily closing prices for the month of January 2010 using a date range. Thomson Reuters Datastream returns closing price data for 21 trading days in the month.

```
price = {'P'};
fromdate = '01/01/2010';
todate = '01/31/2010';

d = fetch(c,sec,price,fromdate,todate)

d = 

  struct with fields:

           Source: 'Datastream'
       Instrument: 'U:IBM~=P~2010-01-01~:2010-01-31'
       StatusType: 'Connected'
       StatusCode: '0'
    StatusMessage: ''
              CCY: 'U$'
             DATE: {21×1 cell}
         DISPNAME: 'INTERNATIONAL BUS.MCHS.'
        FREQUENCY: 'D'
                P: {21×1 cell}
           SYMBOL: 'U:IBM'
```

Access the `DATE` and `P` fields in the structure `d` to display the data.

Retrieve Historical Data for Period

Retrieve monthly closing price data from January 1, 2010, through December 31, 2010.

```
price = {'P'};
fromdate = '01/01/2010';
todate = '12/31/2010';
period = 'M';

d = fetch(c,sec,price,fromdate,todate,period)

d = 

  struct with fields:

           Source: 'Datastream'
       Instrument: 'U:IBM~=P~2010-01-01~:2010-12-31~M'
       StatusType: 'Connected'
       StatusCode: '0'
    StatusMessage: ''
              CCY: 'U$'
             DATE: {12×1 cell}
         DISPNAME: 'INTERNATIONAL BUS.MCHS.'
        FREQUENCY: 'M'
                P: {12×1 cell}
           SYMBOL: 'U:IBM'
```

Determine the highest monthly closing price by converting the character vectors to doubles.

```
IBM = cellfun(@str2num,d.P);
max(IBM)

ans =

        144.41
```

Retrieve Historical Data in Different Currency

Retrieve the monthly closing price for January 2010 in the Euro currency.

```
price = {'P'};
fromdate = '01/01/2010';
todate = '01/31/2010';
period = 'M';
currency = 'EUR';

d = fetch(c,sec,price,fromdate,todate,period,currency)

d = 
```

```
  struct with fields:

           Source: 'Datastream'
       Instrument: 'U:IBM(P)~~EUR~2010-01-01~:2010-01-31~M'
       StatusType: 'Connected'
       StatusCode: '0'
    StatusMessage: ''
              CCY: 'E'
             DATE: '2010-01-01T00:00:00'
         DISPNAME: 'INTERNATIONAL BUS.MCHS. (~E )'
        FREQUENCY: 'M'
                P: '91.2378'
           SYMBOL: 'U:IBM(P)~EUR'
```

Retrieve Historical Data for Multiple Securities

Retrieve weekly closing price data for the IBM and Microsoft securities. Retrieve data from January 1, 2010, through March 31, 2010.

`d` contains a structure array. Each structure in the array contains the data for one security in the security list `sec`.

```
sec = {'U:IBM','@MSFT'};
price = {'P'};
fromdate = '01/01/2010';
todate = '03/31/2010';
period = 'W';

d = fetch(c,sec,price,fromdate,todate,period)

d =

  2×1 struct array with fields:

    Source
    Instrument
    StatusType
    StatusCode
    StatusMessage
    CCY
    DATE
    DISPNAME
    FREQUENCY
    P
```

```
    SYMBOL
```

Access the structure that contains data for the Microsoft security.

```
d(2)
```

```
ans = 

  struct with fields:

           Source: 'Datastream'
       Instrument: '@MSFT~=P~2010-01-01~:2010-03-31~W'
       StatusType: 'Connected'
       StatusCode: '0'
    StatusMessage: ''
              CCY: 'U$'
             DATE: {13×1 cell}
         DISPNAME: 'MICROSOFT'
        FREQUENCY: 'W'
                P: {13×1 cell}
           SYMBOL: '@MSFT'
```

Close Thomson Reuters Datastream Connection

```
close(c)
```

See Also

close | datastream | fetch

Money.Net Topics

Retrieve Current and Historical Money.Net Data

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

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

To access the example code, enter `edit MoneyNetDataWorkflowDemo.m`.

Create Money.Net Connection

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

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

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

Retrieve Money.Net Current Data

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

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

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

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

```
d
```

```
d =

    Symbol      Ask        Bid
    ______    ______    ______

    'IBM'     145.00    143.85
```

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

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

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

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

```
d

d = 

    Symbol      Ask       Bid  
    ______    ______    ______

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

Retrieve Money.Net Historical Data

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

`d` is a table that contains these variables:

- Date timestamp
- High price
- Low price

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

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

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

```
d(1:3,:)

ans =
```

```
       Date              High      Low  
    __________________    ______    ______

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

Determine the average high price in the date range.

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

```
ans =

        170.51
```

Retrieve Money.Net Option Symbol Data

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

```
s = 'IBM';

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

Display the first three option symbols.

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

```
ans =

  3×1 cell array

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

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

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

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

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

d = getdata(c,symbol,f)

d = 

          Symbol                      Description              Strike    Expiration     Ask
    ____________________    ___________________________    ______    __________    ____

    'O:IBM\16F24\131 .0'    'IBM Call 06/24/2016 131.0'    131       06/24/16      23.7
```

Close Money.Net Connection

```
close(c)
```

See Also

close | getdata | moneynet | optionchain | timeseries

More About

- "Retrieve Real-Time Money.Net Data" on page 5-6
- "Retrieve Money.Net News Stories" on page 5-9
- "Money.Net Error and Warning Messages" on page 5-13

External Websites

- Money.Net API Documentation
- Money.Net Help

Retrieve Real-Time Money.Net Data

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

To process real-time data updates, you can use the default event handler function. Or, for a different approach, you can write a custom event handler function. For writing custom event handler functions with Money.Net data, see `realtime`. For custom event handler functions, see "Writing and Running Custom Event Handler Functions" on page 1-35.

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

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

Create Money.Net Connection

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

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

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

Retrieve Real-Time Data for One Symbol

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

```
symbol = 'IBM';

realtime(c,symbol)
```

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

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

Stop Real-Time Data Updates

Stop the symbol subscription.

```
stop(c)
```

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

Retrieve Real-Time Data Using Custom Event Handler Function

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

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

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

```
symbol = 'IBM';
```

```
realtime(c,symbol,myfcn)
```

```
    Symbol            Description               Yesterday    YesterdayDateTime     Bid
    ______    _______________________________    _________    _________________    _____

    'IBM'     'INTERNATIONAL BUSINESS MACHS'    148.31       05/24/16 00:00:00    151.6
```

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

Stop the symbol subscription.

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

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

Close Money.Net Connection

```
close(c)
```

See Also

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

More About

- "Retrieve Current and Historical Money.Net Data" on page 5-2
- "Retrieve Money.Net News Stories" on page 5-9
- "Writing and Running Custom Event Handler Functions" on page 1-35

- "Money.Net Error and Warning Messages" on page 5-13

External Websites

- Money.Net API Documentation
- Money.Net Help

Retrieve Money.Net News Stories

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

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

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

Create Money.Net Connection

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

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

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

Retrieve Specific Number of News Stories

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

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

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

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

```
ans =

                              ArticleTitle                                 ArticleID
    __________________________________________________________________    __________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09
```

Search News Stories Using Search Term

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

```
term = 'Windows';
```

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

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

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

```
ans = 

                              ArticleTitle                                   ArticleID
    ____________________________________________________________________    __________

    'LogMein Shares Edge Lower; LastPass Says Browser Extension Now A…'    4.0005e+09
```

Search News Stories Using Category

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

```
category = 'General Finance';
```

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

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

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

```
ans = 

                              ArticleTitle                                   ArticleID
    ____________________________________________________________________    __________

    'Keep calm and ooze compassion: Leave must seize the moral high g…'    4.0007e+09
```

Search News Stories Using Symbol

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

```
symbol = 'MSFT';
```

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

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

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

```
ans =

                      ArticleTitle                         ArticleID       PublishedTime
    ____________________________________________________    ___________    _________________

    'Microsoft announces after party to Apple's WWDC'    4.0005e+09     06/08/16 12:51:4
```

Analyze News Stories for Analyst Ratings

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

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

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

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

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

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

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

sum(sentiment)

ans =

    33
```

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

Stream News Stories in Real Time

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

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

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

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

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

```
ans = 

                              ArticleTitle                                   ArticleID
    ____________________________________________________________________    __________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09
```

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

Stop the real-time news data stream.

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

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

Close Money.Net Connection

```
close(c)
```

See Also

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

More About

- "Retrieve Current and Historical Money.Net Data" on page 5-2
- "Retrieve Real-Time Money.Net Data" on page 5-6
- "Money.Net Error and Warning Messages" on page 5-13

External Websites

- Money.Net API Documentation
- Money.Net Help

Money.Net Error and Warning Messages

In this section...

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

Money.Net Connection Error Messages

Connection Error Message	Probable Causes	Resolution
Failed to connect to the Money.Net data servers. Confirm that the port number is valid.	The specified port number is invalid. A firewall or proxy is denying the connection.	To specify the default port number, use the first syntax in `moneynet`. To set up a firewall or proxy to work with Money.Net, see Money.Net Firewall Instructions. Your firewall must support a direct internet connection to Money.Net. For details, see Money.Net Firewall Instructions.
Invalid username or password.	The user name and password combination is invalid.	Verify your user name and password. To validate these credentials, contact Money.Net.
User *`username`* is already logged into Money.Net.	A Money.Net connection is open with the specified user name. Only one connection can exist for each user name at a time.	First, to close the previous Money.Net connection, run the `close` function. Then, to create a Money.Net connection, run the `moneynet` function.

Connection Error Message	Probable Causes	Resolution
Lost connection to the Money.Net data server. Use the MONEYNET command to connect again.	The Money.Net data server interrupts the connection. You create multiple `moneynet` objects.	To create a Money.Net connection, run the `moneynet` function.
MONEYNET object is not connected to the Money.Net data servers. Use MONEYNET to create a new connection.	You execute a function using a closed Money.Net connection.	To create a Money.Net connection, run the `moneynet` function.

Money.Net Data Retrieval Error Messages

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

Money.Net Data Retrieval Warning Messages

Data Retrieval Warning Message	Probable Causes	Resolution
No timeseries data returned.	No intraday or historical data that matches the specified input arguments in the `timeseries` function is available. The specified historical date range is outside of the available dates from Money.Net. Second and minute interval data is only available for more recent dates.	Verify that the symbol and interval input arguments are valid. For details, see `timeseries`. Verify that the specified dates contain activity. For example, no data is available on weekends or holidays. Specify a recent date range.
No option month data returned for *`Symbol`*.	The specified symbol is invalid. No options are available for the specified symbol.	Verify the symbol is valid.
No news stories found that match the search criteria.	No news stories are available that match the specified criteria in `news`.	Change the search criteria for news stories. For details, see `news`.
Error occurred while processing real-time data:	The custom event handler function returns an error when processing real-time data.	To find the cause of the error, troubleshoot the custom event handler function code. For writing a custom event handler function using Money.Net data, see `realtime`. For custom event handlers, see "Writing and Running Custom Event Handler Functions" on page 1-35.
Error occurred while processing real-time news:	The custom event handler function returns an error	To find the cause of the error, troubleshoot the

Data Retrieval Warning Message	Probable Causes	Resolution
	when processing real-time news data.	custom event handler function code. For writing a custom event handler function using Money.Net data, see `realtime`. For custom event handlers, see “Writing and Running Custom Event Handler Functions” on page 1-35.

See Also

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

More About

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-6
- “Retrieve Money.Net News Stories” on page 5-9
- “Writing and Running Custom Event Handler Functions” on page 1-35

External Websites

- Money.Net API Documentation
- Money.Net Help

Thomson Reuters Elektron Topics

Decide to Buy Shares Using Thomson Reuters Elektron Current Data

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

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

Create Thomson Reuters Elektron Connection

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

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

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

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

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

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

Retrieve Current Data for Single Security

Retrieve current data for the IBM security.

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

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

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

```
d = 

    FieldId     DataType        Name                       Value
    _______     ________    ____________    _________________________

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

Make Investment Decision to Buy Shares

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

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

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

Close Thomson Reuters Elektron Connection

```
close(c)
```

See Also

`close` | `elektron` | `getdata`

Related Examples

- “Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data” on page 6-5

External Websites

- Thomson Reuters Elektron

Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data

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

The example uses the sample event handler function `elektronExampleListener` to retrieve real-time data for multiple securities in corresponding MATLAB workspace variables. Use this event handler function or write a custom event handler function. For details, see “Writing and Running Custom Event Handler Functions” on page 1-35.

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

Create Thomson Reuters Elektron Connection

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

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

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

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

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

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

Retrieve Real-Time Data for Multiple Securities

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

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

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

```
reqid = 

  struct with fields:

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

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

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

Make Investment Decisions to Buy Shares

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

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

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

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

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

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

Stop Real-Time Data Subscription

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

Close Thomson Reuters Elektron Connection

```
close(c)
```

See Also

`close` | `elektron` | `realtime`

More About

- "Decide to Buy Shares Using Thomson Reuters Elektron Current Data" on page 6-2
- "Writing and Running Custom Event Handler Functions" on page 1-35

External Websites

- Thomson Reuters Elektron

Datafeed Toolbox Graphical User Interface

Retrieve Data Using the Datafeed Dialog Box

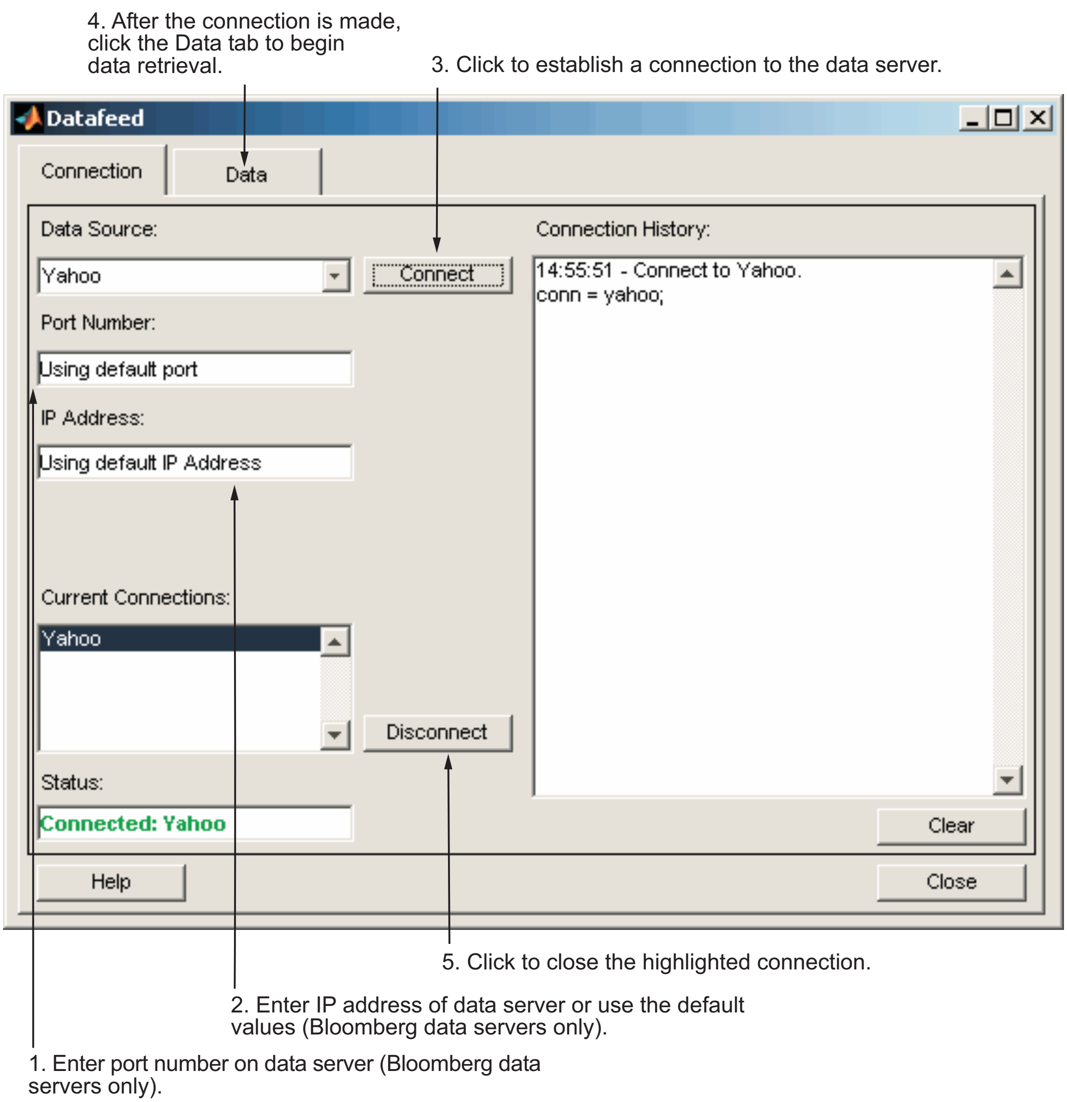
The Datafeed dialog box establishes the connection with the data server and manages data retrieval. Use this dialog box to connect to and retrieve data from Yahoo!

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 7-4.
- The **Data** tab specifies the data request. For details, see "Retrieving Data" on page 7-4.

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



The Datafeed Dialog Box

Connecting to Data Servers

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

2 When the `Connected` message appears in the **Status** field, click the **Data** tab to begin the process of retrieving data from the data server. For details, see "Retrieving Data" on page 7-4.

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

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.

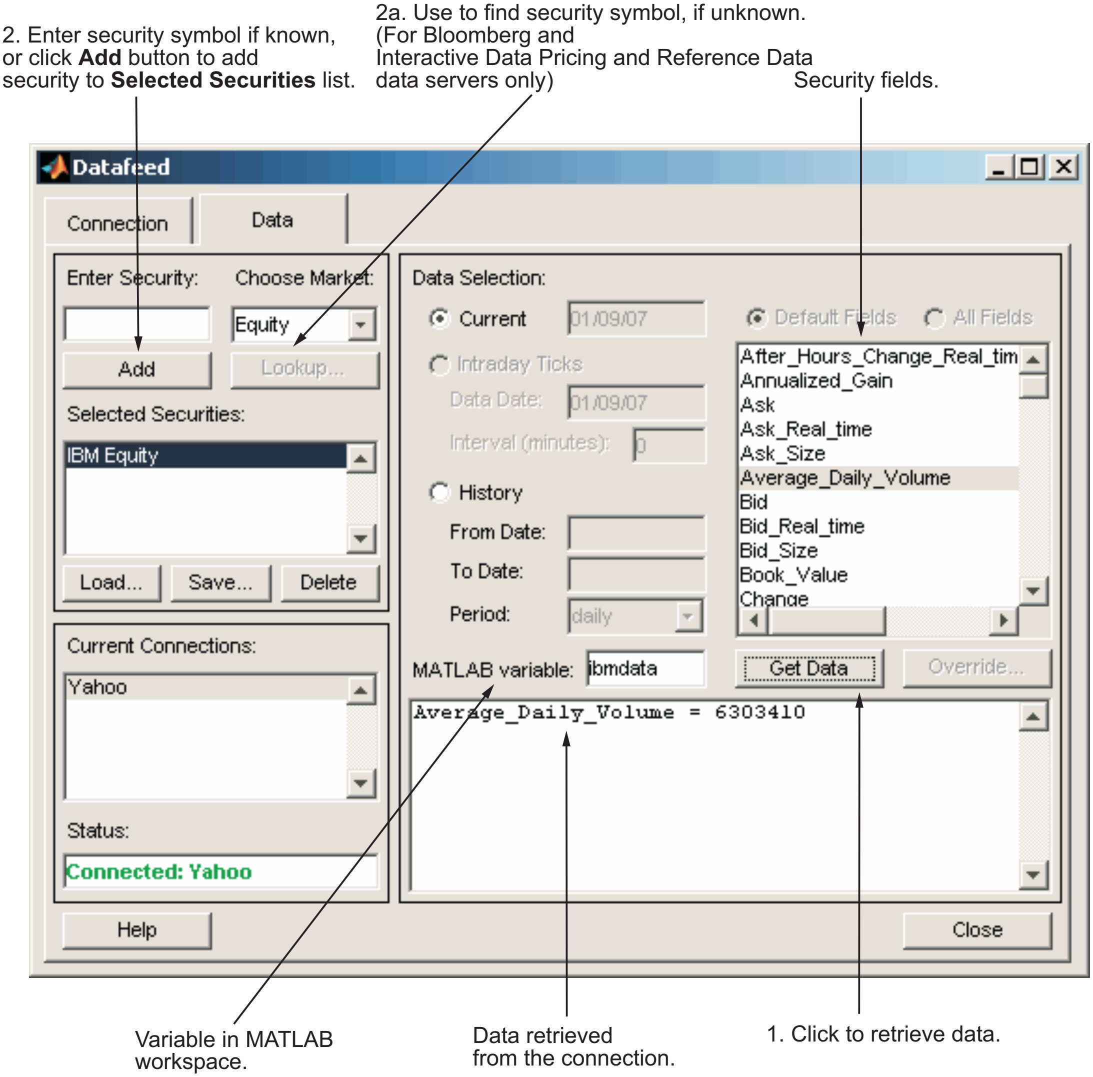
2 Indicate the type of data to retrieve in the **Data Selection** field.

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
Enter Security:
Choose Market:
Equity
Add
Lookup...
Selected Securities:
IBM Equity
Load...
Save...
Delete
Current Connections:
Yahoo
Status:
Connected: Yahoo
Help
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
Get Data
Override...
Average_Daily_Volume = 6303410
Close
Variable in MATLAB workspace.
Data retrieved from the connection.
1. Click to retrieve data.

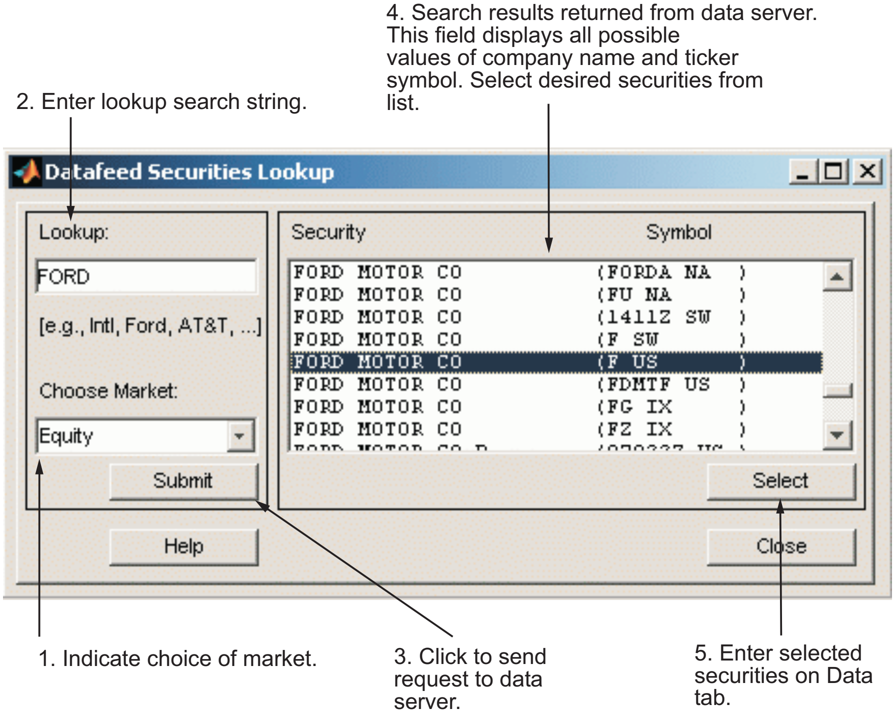
Obtain Ticker Symbol with Datafeed Securities Lookup

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

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

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

The following figure summarizes these steps.


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

8

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 Yahoo! connection.

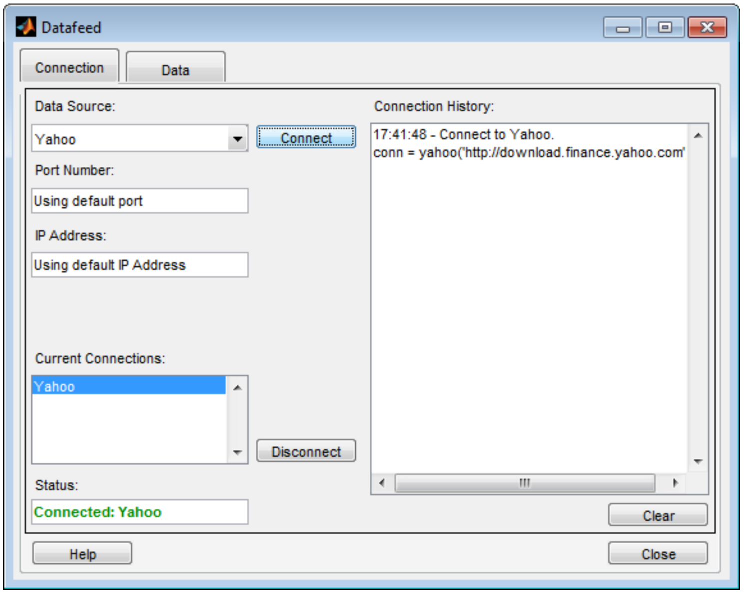
To display this dialog box, enter the `dftool` command in the Command Window.

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 7-4.
- The **Data** tab specifies the data request. For details, see "Retrieving Data" on page 7-4.

Examples

```
dftool
```



See Also

Topics

"Retrieve Data Using the Datafeed Dialog Box" on page 7-2

Introduced in R2013a

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" on page 2-4.

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

Syntax

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

Description

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

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

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

Examples

Connect to Bloomberg Desktop

Create a Bloomberg connection `c` to the 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` with these properties:

- Bloomberg V3 API COM object
- IP address of the local machine
- Port number of the local machine
- Number in milliseconds specifying how long MATLAB attempts to connect to Bloomberg Desktop before timing out

Close the Bloomberg Desktop connection.

```
close(c)
```

Connect to Bloomberg Desktop with a Timeout

Create a Bloomberg 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.

Close the Bloomberg Desktop connection.

```
close(c)
```

- "Connect to Bloomberg" on page 3-2
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19

Input Arguments

`portnumber` — Port number

`[ ]` (default) | numeric scalar

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

Data Types: `double`

`ip` — IP address

`[ ]` (default) | character vector

IP address that identifies the local machine where Bloomberg is running, specified as a character vector.

Data Types: `char`

`timeout` — Timeout value

numeric scalar

Timeout value that denotes the time in milliseconds the local machine attempts to connect before timing out, specified as a numeric scalar.

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

Property	Description
`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

See Also

See Also

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

Topics

"Connect to Bloomberg" on page 3-2
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Data Server Connection Requirements" on page 1-3
"Comparing Bloomberg Connections" on page 2-4
"Workflow for Bloomberg" on page 3-20

Introduced in R2010a

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" on page 2-4.

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

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" on page 3-2

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

character vector

Bloomberg Server IP address, specified as a character vector 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
numeric scalar

Timeout value that denotes the time in milliseconds the local machine attempts to connect before timing out, specified as a numeric scalar.

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

See Also

See Also

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

Topics

“Connect to Bloomberg” on page 3-2
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-4
“Workflow for Bloomberg” on page 3-20

Introduced in R2014b

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” on page 2-4.

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

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” on page 3-2

Input Arguments

`authtype` — Authorization type

`'OS_LOGON'` | `'APPLICATION_ONLY'`

Authorization type, specified as one of these Bloomberg values.

Bloomberg Value	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.

`appname` — Application name
character vector

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

Data Types: `char`

`serverip` — IP address for the machine
character vector | cell array of character vectors

IP address for the machine, specified as a character vector or a cell array of character vectors. A character vector identifies the machine where the Bloomberg B-PIPE process is running, whereas a cell array of character vectors 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
numeric scalar

Timeout value that denotes the time in milliseconds the local machine attempts to connect before timing out, specified as a numeric scalar.

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
`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

See Also

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

Topics
"Connect to Bloomberg" on page 3-2
"Data Server Connection Requirements" on page 1-3
"Comparing Bloomberg Connections" on page 2-4
"Workflow for Bloomberg" on page 3-20

Introduced in R2014b

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" on page 2-4.

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

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

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

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

```
c =

bdl with properties:

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

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

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

```
username = 'xxxxx';
password = 'xxxxxxxx';
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: 'xxxxx'
       Hostname: 'dlsftp.bloomberg.com'
           Port: 30206
     AuthOption: 'key'
        KeyFile: 'c:\temp\mykeyfile'
     Connection: [1x1 com.bloomberg.datalic.api.ExtendedFTPConnection]
```

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

- Bloomberg Data License SFTP server login name
- Bloomberg Data License SFTP server name
- Bloomberg Data License SFTP port number
- Authentication type 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" on page 3-2
- "Retrieve Data Using Bloomberg Data License" on page 3-17

Input Arguments

`username` — User name
character vector

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

Data Types: `char`

`password` — Password
character vector

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

Data Types: `char`

`hostname` — Server name
character vector

Server name, specified as a character vector 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
character vector

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

Data Types: `char`

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

Authentication type, specified as one of the preceding values. 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.

`keyfile` — Key file
character vector

Key file, specified as a character vector 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
character vector

Pass phrase, specified as a character vector. `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

Tips

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

See Also

See Also

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

Topics

“Connect to Bloomberg” on page 3-2
“Retrieve Data Using Bloomberg Data License” on page 3-17
“Data Server Connection Requirements” on page 1-3
“Comparing Bloomberg Connections” on page 2-4
“Workflow for Bloomberg” on page 3-20

Introduced in R2015a

dir

Current Bloomberg Data License connection folder listing

Syntax

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

Description

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

Examples

Request the Bloomberg Data License Folder Listing

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

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

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

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

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

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

```
list = 

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

`list` contains a cell array of character vectors. Each character vector is one folder or file name.

Close the Bloomberg Data License connection.

```
close(c)
```

- “Retrieve Data Using Bloomberg Data License” on page 3-17

Input Arguments

`c` — Bloomberg Data License connection

connection object

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

Output Arguments

`list` — Current working folder contents

cell array of character vectors

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

Tips

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

See Also

See Also

`bdl` | `close`

Topics

“Retrieve Data Using Bloomberg Data License” on page 3-17
“Workflow for Bloomberg” on page 3-20

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 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)
```

- "Retrieve Bloomberg Current Data" on page 3-7
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19

Input Arguments

c — Bloomberg connection

connection object

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

f — Search term

character vector

Search term, specified as a character vector 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.

See Also

See Also

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

Topics

"Retrieve Bloomberg Current Data" on page 3-7
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2010b

close

Close Bloomberg connection V3

Syntax

```
close(c)
```

Description

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

Examples

Close the Bloomberg Connection

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

```
c = blp;
```

Alternatively, you can establish these connections:

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

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

```
close(c)
```


- "Connect to Bloomberg" on page 3-2
- "Retrieve Bloomberg Current Data" on page 3-7
- "Retrieve Bloomberg Historical Data" on page 3-10
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19
- "Retrieve Bloomberg Intraday Tick Data" on page 3-13

- “Retrieve Bloomberg Real-Time Data” on page 3-15

Input Arguments

c — Bloomberg connection

connection object

Bloomberg connection, specified as one of these connection objects:

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

See Also

See Also

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

Topics

“Connect to Bloomberg” on page 3-2
“Retrieve Bloomberg Current Data” on page 3-7
“Retrieve Bloomberg Historical Data” on page 3-10
“Retrieve Current and Historical Data Using Bloomberg” on page 1-19
“Retrieve Bloomberg Intraday Tick Data” on page 3-13
“Retrieve Bloomberg Real-Time Data” on page 3-15
“Workflow for Bloomberg” on page 3-20

Introduced in R2010a

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 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 using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

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

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

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

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

```
ans =

  Columns 1 through 5

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

  Columns 6 through 8

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

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

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

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data for a Screen in German

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

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

```
  Columns 1 through 5

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

  Columns 6 through 8

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

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

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

- Earnings per share

Close the connection.

```
close(c)
```

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

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

```
ans =

  Columns 1 through 5

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

  Columns 6 through 8

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

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

- Ticker symbol
- Company name
- Market capitalization
- Price

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

Close the connection.

```
close(c)
```

Retrieve Equity Screening Data Using Override Fields

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

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

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

```
ans = 

  Columns 1 through 5

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

  Columns 6 through 8
```

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

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

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

Close the connection.

```
close(c)
```

- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19

Input Arguments

`c` — Bloomberg connection

connection object

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

`sname` — Screen name

character vector

Screen name, specified as a character vector 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 preceding values to denote the Bloomberg screen type. `'GLOBAL'` denotes a Bloomberg screen name and `'PRIVATE'` denotes a customized screen name. When using the optional `group` input argument, `stype` cannot be set to `'PRIVATE'` for customized screen names.

`languageid` — Language identifier
character vector

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

Data Types: `char`

`group` — Group identifier
character vector

Group identifier, specified as a character vector 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.

See Also

See Also
`blp` | `close` | `getdata` | `tahistory`

Topics
"Retrieve Bloomberg Current Data" on page 3-7
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2012b

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 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)
```


- "Retrieve Bloomberg Current Data" on page 3-7
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19

Input Arguments

`c` — Bloomberg connection

connection object

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

`f` — Field mnemonic

character vector

Field mnemonic, specified as a character vector that denotes the Bloomberg field information to retrieve.

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.

See Also

See Also

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

Topics

“Retrieve Bloomberg Current Data” on page 3-7
“Retrieve Current and Historical Data Using Bloomberg” on page 1-19
“Workflow for Bloomberg” on page 3-20

Introduced in R2010b

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 term `f`.

Examples

Search for the Bloomberg Last Price Field

Create a Bloomberg connection.

```
c = blp;
```

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

Return data for the search term `'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)
```

- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19

Input Arguments

c — Bloomberg connection

connection object

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

f — Search term

character vector

Search term, specified as a character vector that denotes the Bloomberg field descriptive data to retrieve.

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.

See Also

See Also

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

Topics

"Retrieve Bloomberg Current Data" on page 3-7
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2010b

get

Properties of Bloomberg connection V3

Syntax

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

Description

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

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

Examples

Retrieve Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the Bloomberg connection properties.

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

```
v = 

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

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

Close the Bloomberg connection.

```
close(c)
```

Retrieve One Bloomberg Connection Property

Create the Bloomberg connection.

```
c = blp;
```

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

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

```
property = 'port';
v = get(c,property)
```

```
v =

        8194
```

`v` is a double that contains the port number of the Bloomberg connection object.

Close the Bloomberg connection.

```
close(c)
```

Retrieve Two Bloomberg Connection Properties

Create the Bloomberg connection.

```
c = blp;
```

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

Create a cell array `properties` with character vectors `'session'` and `'port'`. Retrieve the Bloomberg session object and port number from the Bloomberg connection object.

```
properties = {'session','port'};
v = get(c,properties)

v = 

    session: [1x1 com.bloomberglp.blpapi.Session]
       port: 8194
```

`v` is a structure containing the Bloomberg session object and port number.

Close the Bloomberg connection.

```
close(c)
```

- "Connect to Bloomberg" on page 3-2

Input Arguments

`c` — Bloomberg connection

connection object

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

`properties` — Property names

character vector | cell array of character vectors

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

Data Types: `char` | `cell`

Output Arguments

`v` — Bloomberg connection properties

numeric scalar | character vector | object | structure

Bloomberg connection properties, returned as these data types depending on the requested properties.

Requested Properties	Data Type
Port number or timeout	Numeric scalar
IP address	Character vector
Bloomberg session	Object
All properties	Structure

See Also

See Also

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

Topics

"Connect to Bloomberg" on page 3-2
"Workflow for Bloomberg" on page 3-20

Introduced in R2010a

getbulkdata

Bulk data with header information for Bloomberg connection V3

Syntax

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

Description

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

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

`d = getbulkdata(c,s,f,o,ov,Name,Value)` returns the bulk data with additional options specified by one or more `Name,Value` pair arguments for 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 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 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 using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

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

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

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

d = 

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

sec = 

    'IBM US Equity'
```

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

Display the closing price.

```
d.LAST_PRICE
```

```
ans =

    '188.74'
```

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

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

```
ans =

  Columns 1 through 6

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

  Column 7

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

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

Close the connection.

```
close(c)
```

- "Retrieve Bloomberg Current Data" on page 3-7
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19

Input Arguments

c — Bloomberg connection

connection object

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

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors 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

character vector | cell array of character vectors

Bloomberg data fields, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg data field name. A cell array of character vectors denotes multiple Bloomberg data field names. For details about the character vectors 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

character vector | cell array of character vectors

Bloomberg override field, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg override field name. A cell array of character vectors denotes multiple Bloomberg override field names. For details about the character vectors 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

character vector | cell array of character vectors

Bloomberg override field value, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg override field value. A cell array of character vectors denotes multiple Bloomberg override field values. 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 the data type character vector.

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 of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`
- `svm`
- `ticker` (default)
- `wpk`

See Also

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

Topics

“Retrieve Bloomberg Current Data” on page 3-7
“Retrieve Current and Historical Data Using Bloomberg” on page 1-19
“Workflow for Bloomberg” on page 3-20

Introduced in R2014b

getdata

Current data for Bloomberg connection V3

Syntax

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

Description

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

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

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

`[d,sec] = getdata(___)` additionally returns the security list `sec` using any of the input 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 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 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 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 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 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" on page 3-7
- "Retrieve Current and Historical Data Using Bloomberg" on page 1-19

Input Arguments

c — Bloomberg connection
connection object

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

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors 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

character vector | cell array of character vectors

Bloomberg data fields, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg data field name. A cell array of character vectors denotes multiple Bloomberg data field names. For details about the character vectors 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

character vector | cell array of character vectors

Bloomberg override field, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg override field name. A cell array of character vectors denotes multiple Bloomberg override field names. For details about the character vectors 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

character vector | cell array of character vectors

Bloomberg override field value, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg override field value. A cell array of character vectors denotes multiple Bloomberg override field values. 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 the data type character vector.

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 of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`
- `svm`
- `ticker` (default)
- `wpk`

Tips

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

- You can check data and field availability by using the Bloomberg Excel® Add-In.
- For a Bloomberg B-PIPE connection, `d` returns an additional field named `EID`. `EID` means entitlement identifier. For details, see the *Bloomberg API Developer's Guide*.

See Also

See Also

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

Topics

"Retrieve Bloomberg Current Data" on page 3-7
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2010a

history

Historical data for Bloomberg connection V3

Syntax

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

Description

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

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

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

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

`[d,sec] = history( ___ )` additionally returns the security list `sec` using any of the input 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 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 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 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 using blpsrv or Bloomberg B-PIPE using bpipe.

Get the monthly closing price from August 1, 2010 through August 1, 2011 for the IBM security in U.S. currency 'USD'. The period values 'monthly', 'actual', and 'all_calendar_days' specify returning actual monthly data for all calendar days. The period value 'nil_value' specifies filling missing data values with a NaN.

```
[d,sec] = history(c,'IBM US Equity','LAST_PRICE',...
                  '8/01/2010','8/01/2011',{'monthly','actual',...
                  'all_calendar_days','nil_value'},'USD')

d =

    734351.00        128.40
    734382.00        125.77
    734412.00        135.64
    734443.00        143.32
    734473.00        144.41
```

```
    734504.00         146.76
    734535.00         163.56
    734563.00         159.97
    734594.00         164.27
    734624.00         170.58
    734655.00         166.56
    734685.00         174.54
    734716.00         180.75


sec =

    'IBM US Equity'
```

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

Close the Bloomberg connection.

```
close(c)
```

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 using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

Get the weekly 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 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 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 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 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 using blpsrv or Bloomberg B-PIPE using bpipe.

Retrieve the median estimated earnings per share for AkzoNobel® from October 1, 2010 through October 30, 2010. When specifying Bloomberg override fields, use the character vector 'overrideFields'. The overrideFields argument must be an n-by-2 cell array, where the first column is the override field and the second column is the override value.

```
d = history(c,'AKZA NA Equity', ...
            'BEST_EPS_MEDIAN',datenum('01.10.2010', ...
            'dd.mm.yyyy'),datenum('30.10.2010','dd.mm.yyyy'), ...
            {'daily','calendar'},[],'overrideFields', ...
            {'BEST_FPERIOD_OVERRIDE','BF'})

d =

    734412.00          3.75
    734415.00          3.75
    734416.00          3.75
    ...
```

d returns the numeric representation for the date in the first column and the median estimated earnings per share in the second column.

Close the Bloomberg connection.

```
close(c)
```

- “Retrieve Bloomberg Historical Data” on page 3-10
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19

Input Arguments

`c` — Bloomberg connection

connection object

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

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors 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

character vector | cell array of character vectors

Bloomberg data fields, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg data field name. A cell array of character vectors denotes multiple Bloomberg data field names. For details about the character vectors 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'` | ...

Periodicity, specified as one of these values to denote the data to return. For specifying multiple values, use a cell array. For example, when `period` is set to {'daily','all_calendar_days'}, history returns daily data for all calendar days, and reports missing data as NaNs. When period is set to 'active_days_only', history returns data using the default periodicity for active trading days only. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. These tables show the values for period.

To specify the periodicity of the return data, see this table.

Value	Description
'daily'	Return data for each day.
'weekly'	Return data for each week.
'monthly'	Return data for each month.
'quarterly'	Return data for each quarter.
'semi_annually'	Return data semiannually.
'yearly'	Return data for each year.

The anchor date is the date to which all other reported dates are related. To specify the anchor date, see this table.

Value	Description
'actual'	Anchor date specification for an actual date. For this function, for periodicities other than daily, enddate is the anchor date. If the period is weekly and the enddate is a Thursday, every data point is a Thursday, or the nearest prior business day to Thursday. If the period is monthly and the enddate is the 20th of a month, every data point is the 20th of each month in the date range.
'calendar'	Anchor date specification for a calendar year.
'fiscal'	Anchor date specification for a fiscal year.
'none'	Do not specify the anchor date.

To specify returning data for particular days, see this table.

Value	Description
`'non_trading_weekdays'`	Return data for all weekdays.
`'all_calendar_days'`	Return data for all calendar days.
`'active_days_only'`	Return data for only active trading days.

To specify how to fill missing values, see this table.

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

Data Types: `char` | `cell`

`currency` — Currency
character vector

Currency, specified as a character vector 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 | character vector | cell array of character vectors

Beginning date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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 | character vector | cell array of character vectors

End date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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 of character vectors

Security list, returned as a cell array of character vectors for the corresponding securities in `s`. The contents of `sec` are identical in value and order to `s`. You can return securities with any of the following identifiers:

- `buid`
- `cats`
- `cins`
- `common`
- `cusip`
- `isin`
- `sedol1`
- `sedol2`
- `sicovam`
- `svm`
- `ticker` (default)
- `wpk`

Tips

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

See Also

See Also
`blp` | `close` | `getdata` | `realtime` | `timeseries`

Topics
“Retrieve Bloomberg Historical Data” on page 3-10

"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2010a

isconnection

Determine Bloomberg connection V3

Syntax

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

Description

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

Examples

Validate the Bloomberg Connection

Create the Bloomberg connection.

```
c = blp;
```

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

Validate the Bloomberg connection.

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

```
v =

     1
```

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

Close the Bloomberg connection.

```
close(c)
```

- “Connect to Bloomberg” on page 3-2

Input Arguments

c — Bloomberg connection

connection object

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

Output Arguments

v — Valid Bloomberg connection

true | false

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

See Also

See Also

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

Topics

"Connect to Bloomberg" on page 3-2
"Workflow for Bloomberg" on page 3-20

Introduced in R2010b

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 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 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 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)
```

- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19

Input Arguments

c — Bloomberg connection

connection object

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

q — Keyword query

character vector | cell array of character vectors

Keyword query, specified as a character vector or a cell array of character vectors. Each character vector denotes an item for which information is requested. For example, the keyword query can be a security, a curve type, or a filter ticker.

Data Types: `char`

`reqtype` — Request type

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

Request type, specified as the preceding values to denote the type of information request. `'instrumentListRequest'` denotes a security or instrument lookup request. `'curveListRequest'` denotes a curve lookup request. `'govtListRequest'` denotes a government lookup request for government securities.

Name-Value Pair Arguments

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

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

Data Types: `char`

`'languageOverride'` — Language override
character vector

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

Data Types: `char`

`'countryCode'` — Country code
character vector

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

Data Types: `char`

`'currencyCode'` — Currency code
character vector

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

Data Types: `char`

`'curveID'` — Bloomberg short-form identifier for curve
character vector

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

Data Types: `char`

`'type'` — Bloomberg market sector type
character vector

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

Data Types: `char`

`'subtype'` — Bloomberg market sector subtype
character vector

Bloomberg market sector subtype, specified as a character vector 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 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

Property	Description
`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

See Also

See Also

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

Topics

"Retrieve Bloomberg Current Data" on page 3-7

“Retrieve Current and Historical Data Using Bloomberg” on page 1-19
“Workflow for Bloomberg” on page 3-20

Introduced in R2014a

portfolio

Current portfolio data for Bloomberg connection V3

Syntax

```
d = portfolio(c,p,f)
d = portfolio(c,p,f,o,ov)
[d,plist] = portfolio( ___ )
```

Description

`d = portfolio(c,p,f)` returns current portfolio data for the fields `f` in the portfolio `p` using the Bloomberg connection `c`.

`d = portfolio(c,p,f,o,ov)` returns current portfolio data using override field `o` and override value `ov`.

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

Examples

Request Portfolio Data

Create the Bloomberg connection.

```
c = blp;
```

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

Request portfolio data for a custom portfolio with portfolio identifier `U335877-1 Client`. Request data using all fields `f`.

```
p = 'U335877-1 Client';
f = {'PORTFOLIO_MEMBERS','PORTFOLIO_MPOSITION',...
```

```
     'PORTFOLIO_MWEIGHT','PORTFOLIO_DATA'};

d = portfolio(c,p,f)

d = 

    PORTFOLIO_MPOSITION: {{0x1 cell}}
      PORTFOLIO_MWEIGHT: {{0x1 cell}}
         PORTFOLIO_DATA: {{0x1 cell}}
      PORTFOLIO_MEMBERS: {{0x1 cell}}
```

`d` is a structure that contains portfolio data. Each structure field corresponds to data for each portfolio field.

Close the connection.

```
close(c)
```

Request Portfolio Data Using a Specific Date

Create the Bloomberg connection.

```
c = blp;
```

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

Request portfolio data for a custom portfolio with portfolio identifier `U335877-1 Client`. Request data using all fields `f`. Filter the portfolio data by specifying the date of November 3, 2014 using the override value `REFERENCE_DATE` equal to `20141103`.

```
p = 'U335877-1 Client';
f = {'PORTFOLIO_MEMBERS','PORTFOLIO_MPOSITION',...
     'PORTFOLIO_MWEIGHT','PORTFOLIO_DATA'};
o = {'REFERENCE_DATE'};
ov = {'20141103'};

[d,plist] = portfolio(c,p,f,o,ov)

d = 

    PORTFOLIO_MPOSITION: {{0x1 cell}}
      PORTFOLIO_MWEIGHT: {{0x1 cell}}
         PORTFOLIO_DATA: {{0x1 cell}}
      PORTFOLIO_MEMBERS: {{0x1 cell}}
```

```
plist =

    'U335877-1 Client'
```

d is a structure that contains portfolio data. Each structure field corresponds to data for each portfolio field.

plist is a cell array that contains the portfolio identifier.

Close the connection.

```
close(c)
```

- “Retrieve Bloomberg Current Data” on page 3-7

Input Arguments

c — Bloomberg connection

connection object

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

p — Portfolio list

character vector | cell array of character vectors

Portfolio list, specified as a character vector for one portfolio or a cell array of character vectors for multiple portfolios. Specify the portfolio by the portfolio identifier that you can find in the upper-right corner of the settings tab on the portfolio page. Access the portfolio page by using the **PRTU<GO>** option from the Bloomberg terminal. For details, see the *Bloomberg API Developer’s Guide* using the **WAPI <GO>** option from the Bloomberg terminal.

Data Types: char | cell

f — Portfolio fields

'PORTFOLIO_DATA' | 'PORTFOLIO_MEMBERS' | 'PORTFOLIO_MPOSITION' | 'PORTFOLIO_MWEIGHT'

Portfolio fields, specified as one of the preceding values for one field. To specify multiple fields, use a cell array of these values.

Bloomberg Field Name	Bloomberg Field Description
'PORTFOLIO_DATA'	Returns a list of the identifiers, positions, market values, cost, cost date, and cost foreign exchange rate of each security in a custom portfolio.
'PORTFOLIO_MEMBERS'	Returns a list of identifiers for the members of a custom portfolio.
'PORTFOLIO_MPOSITION'	Returns a list of identifiers and the position for each security in a custom portfolio.
'PORTFOLIO_MWEIGHT'	Returns a list of identifiers and the percentage weight for each security in a custom portfolio.

Data Types: `char` | `cell`

o — Bloomberg override field

character vector | cell array of character vectors

Bloomberg override field, specified as a character vector or cell array of character vectors. The Bloomberg value `'REFERENCE_DATE'` denotes returning Bloomberg data for a specific date.

Data Types: `char` | `cell`

ov — Bloomberg override field value

character vector | cell array of character vectors

Bloomberg override field value, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg override field value. A cell array of character vectors denotes multiple Bloomberg override field values. Use this field value to filter the Bloomberg data result set.

Example: `'20100101'`

Data Types: `char` | `cell`

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.

`plist` — Portfolio list
cell array of character vectors

Portfolio list, returned as a cell array of character vectors for the corresponding portfolio identifiers in `p`. The contents of `plist` are identical in value and order to `p`.

See Also

See Also

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

Topics

"Retrieve Bloomberg Current Data" on page 3-7
"Workflow for Bloomberg" on page 3-20

Introduced in R2015b

realtime

Real-time data for Bloomberg connection V3

Syntax

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

Description

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

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

Examples

Retrieve Data for One Security

Retrieve a snapshot of data for one security only.

Create the Bloomberg connection.

```
c = blp;
```

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

Retrieve the last trade and volume of the IBM security.

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

```
d =

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

Close the Bloomberg connection.

```
close(c)
```

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

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

```
subs =

com.bloomberglp.blpapi.SubscriptionList@79f07684

   Timer Object: timer-2

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

   Callbacks
```

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

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

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

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

Close the Bloomberg connection.

```
close(c)
```

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

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

Create the Bloomberg connection.

```
c = blp;
```

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

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

`v3stockticker` requires the input argument `f` of `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 using `blpsrv` or Bloomberg B-PIPE using `bpipe`.

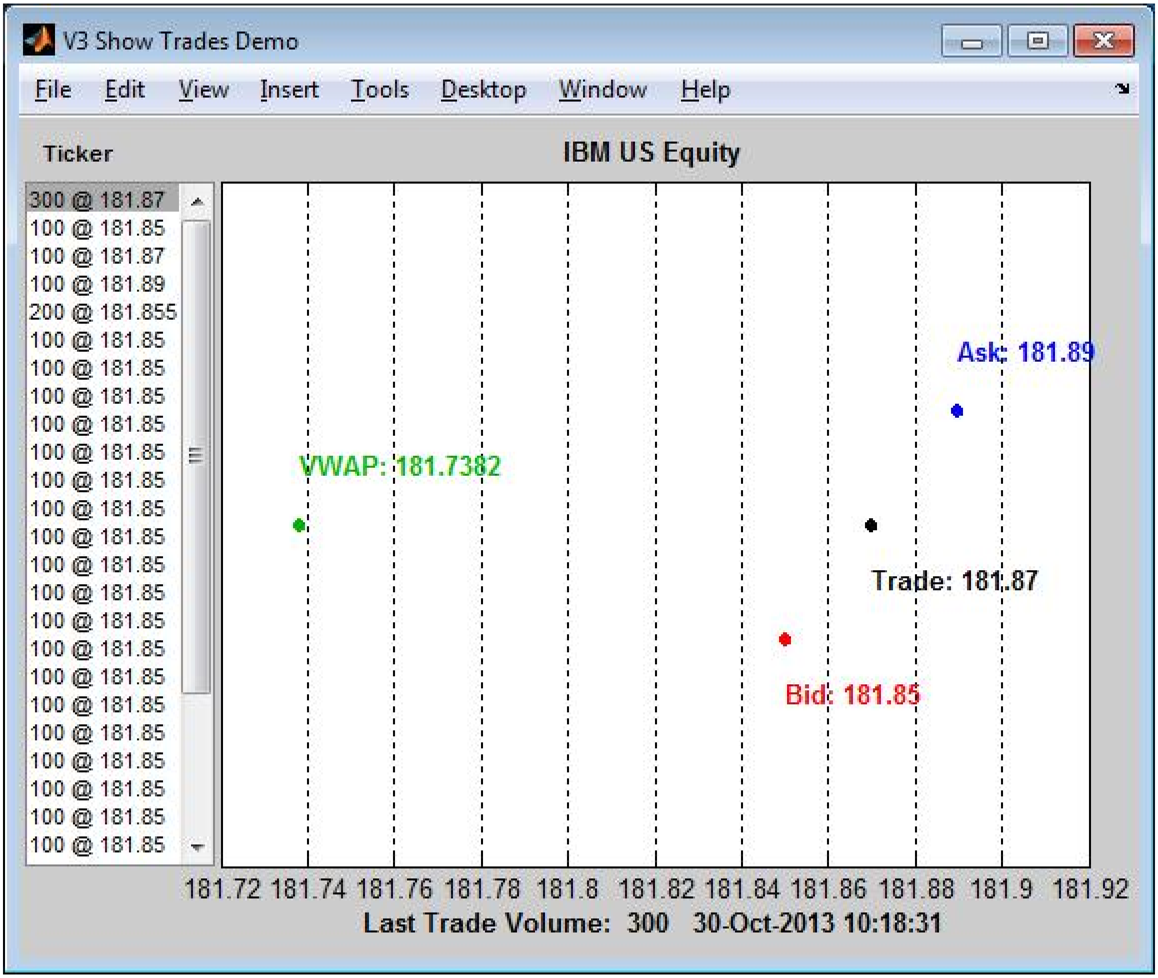
Retrieve volume, last trade, bid, ask, and volume weight adjusted price (VWAP) data for the IBM security using the event handler `v3showtrades`.

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

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

```
subs =

com.bloomberglp.blpapi.SubscriptionList@5c17dcdb


   Timer Object: timer-4

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

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

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



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

Close the Bloomberg connection.

```
close(c)
```

Retrieve Data for One Security Using the Event Handler v3pricevol

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

Create the Bloomberg connection.

```
c = blp;
```

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

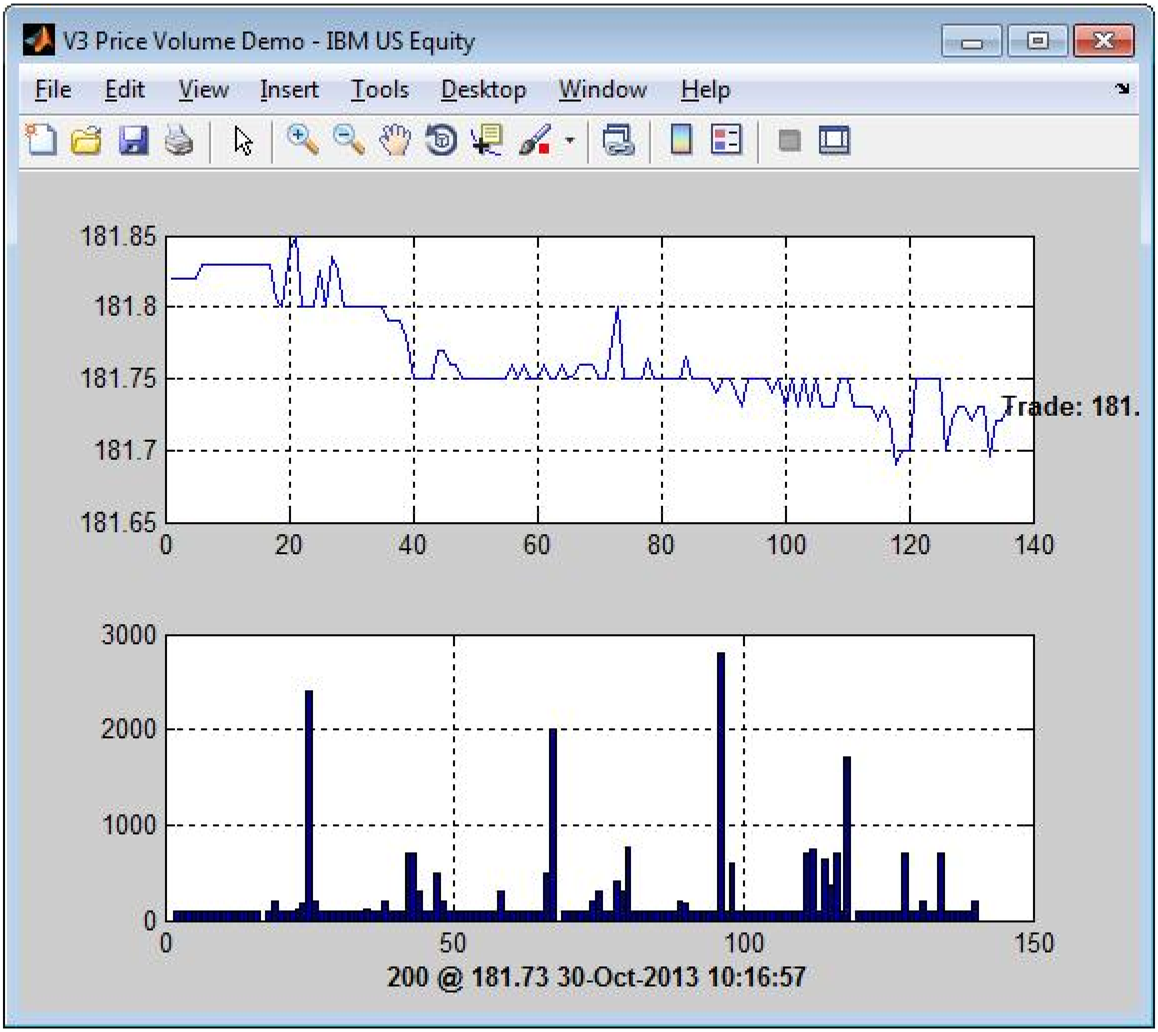
Retrieve last price and volume data for the IBM security using event handler `v3pricevol`.

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

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

```
subs =

com.bloomberglp.blpapi.SubscriptionList@16f66676


   Timer Object: timer-5

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

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

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



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

Close the Bloomberg connection.

```
close(c)
```

- "Retrieve Bloomberg Real-Time Data" on page 3-15

Input Arguments

`c` — Bloomberg connection

connection object

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

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors 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

character vector | cell array of character vectors

Bloomberg data fields, specified as a character vector or a cell array of character vectors. A character vector denotes one Bloomberg data field name. A cell array of character vectors denotes multiple Bloomberg data field names. For details about the character vectors 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

character vector

Event handler, specified as a character vector that denotes the name of an event handler function that you define. You can define an event handler function to process any type of real-time Bloomberg events. The specified event handler function runs every time the timer fires.

Data Types: `char`

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.

See Also

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

Topics
"Retrieve Bloomberg Real-Time Data" on page 3-15
"Workflow for Bloomberg" on page 3-20
"Writing and Running Custom Event Handler Functions" on page 1-35

Introduced in R2010a

stop

Unsubscribe real-time requests for Bloomberg connection V3

Syntax

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

Description

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

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

Examples

Stop Real-Time Requests

Unsubscribe to real-time data for one security.

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

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

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

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

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

Close the Bloomberg connection.

```
close(c)
```

Stop Real-Time Requests for a Security List

Create the Bloomberg connection.

```
c = blp;
```

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

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

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

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

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

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

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

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

Close the Bloomberg connection.

```
close(c)
```

- "Retrieve Bloomberg Real-Time Data" on page 3-15

Input Arguments

`c` — Bloomberg connection

connection object

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

`subs` — Bloomberg subscription

object

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

`t` — MATLAB timer

object

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

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors for multiple securities. You can specify the security by name or by CUSIP, and with or without the pricing source.

Data Types: `char` | `cell`

See Also

See Also

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

Topics

"Retrieve Bloomberg Real-Time Data" on page 3-15
"Workflow for Bloomberg" on page 3-20

Introduced in R2010a

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 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 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)
```

- “Retrieve Bloomberg Current Data” on page 3-7
- “Retrieve Current and Historical Data Using Bloomberg” on page 1-19

Input Arguments

c — Bloomberg connection

connection object

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

s — Security

character vector

Security, specified as a character vector for a single Bloomberg security.

Data Types: `char`

`startdate` — Start date

numeric scalar | character vector

Start date, specified as a numeric scalar or character vector 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
numeric scalar | character vector

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

Example: `floor(now)`

Data Types: `double` | `char`

`study` — Study type
character vector

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

Data Types: `char`

`period` — Periodicity
`'daily'` | `'weekly'` | `'monthly'` | `'quarterly'` | ...

Periodicity, specified as one of these values to denote the data to return. For specifying multiple values, use a cell array. For example, when `period` is set to `{'daily','all_calendar_days'}`, `tahistory` returns daily data for all calendar days, and reports missing data as `NaN`s. When `period` is set to `'active_days_only'`, `tahistory` returns data using the default periodicity for active trading days only. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. These tables show the values for `period`.

To specify the periodicity of the return data, see this table.

Value	Description
`'daily'`	Return data for each day.
`'weekly'`	Return data for each week.
`'monthly'`	Return data for each month.
`'quarterly'`	Return data for each quarter.
`'semi_annually'`	Return data semiannually.
`'yearly'`	Return data for each year.

The anchor date is the date to which all other reported dates are related. To specify the anchor date, see this table.

Value	Description
`'actual'`	Anchor date specification for an actual date. For this function, for periodicities other than daily, `enddate` is the anchor date. If the period is weekly and the `enddate` is a Thursday, every data point is a Thursday, or the nearest prior business day to Thursday. If the period is monthly and the `enddate` is the 20th of a month, every data point is the 20th of each month in the date range.
`'calendar'`	Anchor date specification for a calendar year.
`'fiscal'`	Anchor date specification for a fiscal year.

To specify returning data for particular days, see this table.

Value	Description
`'non_trading_weekdays'`	Return data for all weekdays.
`'all_calendar_days'`	Return data for all calendar days.
`'active_days_only'`	Return data for only active trading days.

To specify how to fill missing values, see this table.

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

Data Types: `char` | `cell`

Name-Value Pair Arguments

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

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

Data Types: `char`

`'priceSourceLow'` — Low price
character vector

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

Data Types: `char`

`'priceSourceClose'` — Closing price
character vector

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

Data Types: `char`

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.

See Also

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

Topics
"Retrieve Bloomberg Current Data" on page 3-7
"Retrieve Current and Historical Data Using Bloomberg" on page 1-19
"Workflow for Bloomberg" on page 3-20

Introduced in R2012b

timeseries

Intraday tick data for Bloomberg connection V3

Syntax

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

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

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 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 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 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 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 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 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 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 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" on page 3-13

Input Arguments

c — Bloomberg connection
connection object

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

s — Security
character vector

Security, specified as a character vector for a single Bloomberg security.

Data Types: `char`

date — Date
numeric scalar | character vector

Date, specified as a numeric scalar or character vector 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
`'TRADE'` | `'BID'` | `'ASK'` | ...

Bloomberg field, specified as one of these values that define the tick data to return.

Request Type	Valid Bloomberg Field Values
IntradayBarRequest with time interval specified	`'TRADE'`
	`'BID'`

Request Type	Valid Bloomberg Field Values
	'ASK'
	'BID_BEST'
	'ASK_BEST'
IntradayTickRequest with no time interval specified	'TRADE'
	'BID'
	'ASK'
	'BID_BEST'
	'ASK_BEST'
	'SETTLE'

`options` — Bloomberg API options
`'includeConditionCodes' | 'includeExchangeCodes' | 'includeBrokerCodes' | ...`

Bloomberg API options, specified as one of these values.

Value	Description
'includeConditionCodes'	Exchange condition codes associated with the event
'includeExchangeCodes'	Exchange code where tick originated
'includeBrokerCodes'	Broker code
'includeRpsCodes'	Reporting party side
'includeNonPlottableEvents'	After-hours data

To specify more than one Bloomberg API option, use a cell array of these values.

Specify the corresponding Bloomberg API value for each API option.

For example, to specify one Bloomberg API option, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
              'includeConditionCodes','true');
```

To specify two Bloomberg API options, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
              {'includeConditionCodes','includeExchangeCodes'},...
              {'true','true'});
```

Data Types: `char` | `cell`

`values` — Bloomberg API values

`'true'` | `'false'`

Bloomberg API values, specified as `'true'` or `'false'`. Each value corresponds to the specified Bloomberg API option. To specify more than one Bloomberg API value, use a cell array.

For example, to specify one Bloomberg API option, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
              'includeConditionCodes','true');
```

To specify two Bloomberg API options, enter:

```
d = timeseries(c,'F US Equity',floor(now),[],'Trade',...
              {'includeConditionCodes','includeExchangeCodes'},...
              {'true','true'});
```

Data Types: `char` | `cell`

`startdate` — Start date

numeric scalar | character vector

Start date, specified as a numeric scalar or character vector 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

numeric scalar | character vector

End date, specified as a numeric scalar or character vector 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`.

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” (MATLAB).
- You cannot retrieve Bloomberg intraday tick data for a date more than 140 days ago.
- The *Bloomberg API Developer’s Guide* states that `'TRADE'` corresponds to LAST_PRICE for IntradayTickRequest and IntradayBarRequest.

- Bloomberg V3 intraday tick data supports additional name-value pairs. For details on these pairs, see the *Bloomberg API Developer’s Guide* by typing `WAPI` and clicking the **<GO>** button on the Bloomberg terminal.
- You can check data and field availability by using the Bloomberg Excel Add-In.

See Also

See Also

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

Topics

“Retrieve Bloomberg Intraday Tick Data” on page 3-13
“Workflow for Bloomberg” on page 3-20

Introduced in R2010a

datastream

Establish connections to Thomson Reuters Datastream API

Syntax

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

Arguments

`'UserName'`	Thomson Reuters user name, specified as a character vector that must start with `DS:`.
`'Password'`	Thomson Reuters password, specified as a character vector.
`'Source'`	To connect to the Thomson Reuters Datastream API, enter the character vector `'Datastream'`.
`'URL'`	Thomson Reuters web URL, specified as a character vector.

Description

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

Examples

Connect to the Thomson Reuters Datastream API using these sample values:

- User name `'DS:User1'`
- Password `'Pass1'`
- URL `'http://dataworks.thomson.com/Dataworks/Enterprise/1.0'`

To retrieve your values for these input arguments, contact Thomson Reuters.

```
Connect = datastream('DS: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

See Also

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

Introduced in R2006a

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

See Also

`datastream`

Introduced in R2006a

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 character vector containing the name of a security, or cell array of character vectors containing names of multiple securities. This data is in a format recognizable by the Thomson Reuters Datastream data server.
`'Fields'`	(Optional) MATLAB character vector or cell array of character vectors indicating the data fields for which to retrieve data.
`'Date'`	(Optional) MATLAB character vector 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 character vectors or empty arrays ([ ]).

Description

data = fetch(Connect, 'Security') returns the default time series for the indicated security.

data = fetch(Connect, 'Security', 'Fields') returns data for the specified security and fields.

data = fetch(Connect, 'Security', 'Fields', 'Date') returns data for the specified security and fields on a particular date.

data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate') returns data for the specified security and fields for the indicated date range.

data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period') returns instrument data for the given range with the indicated period.

data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency') also specifies the currency in which to report the data.

data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency', 'ReqFlag') also specifies a ReqFlag that determines how the request is processed by Datastream.

Note: The Thomson Reuters Datastream interface returns all data as character vectors. For example, it returns Price data to the MATLAB workspace as a cell array of character vectors within the structure. There is no way to determine the data type from the Datastream interface. For details about the Thomson Reuters Datastream returned data, see Reuters Data Support.

Examples

Retrieving Time-Series Data

Return the trailing one-year price time series for the instrument ICI, with the default value P for the 'Fields' argument using the command:

```
data = fetch(Connect, 'ICI')
```

Or the command:

```
data = fetch(Connect, 'ICI', 'P')
```

Retrieving Opening and Closing Prices

Return the closing and opening prices for the instruments ICI on the date September 1, 2007.

```
data = fetch(Connect, 'ICI', {'P', 'PO'}, '09/01/2007')
```

Retrieving Monthly Opening and Closing Prices for a Specified Date Range

Return the monthly closing and opening prices for the securities ICI and IBM from 09/01/2005 to 09/01/2007:

```
data = fetch(Connect, {'ICI', 'IBM'}, {'P', 'PO'}, ...
'09/01/2005', '09/01/2007', 'M')
```

Retrieving Static Data

Return the static fields NAME and ISIN:

```
data = fetch(Connect,{'IBM~REP'}, {'NAME','ISIN'});
```

You can also return SECD in this way.

Retrieving Russell 1000 Constituent List

Return the Russell 1000 Constituent List:

```
russell = fetch(Connect, {'LFRUSS1L~LIST~#UserName'});
```

where UserName is the user name for the Thomson Reuters Datastream connection.

See Also

See Also

close | datastream | get | isconnection

Introduced in R2006a

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 character vector or cell array of character vectors 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

See Also

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

Introduced in R2006a

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

See Also

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

Introduced in R2006a

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

See Also

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

Topics

"Retrieve Intraday and Historical Data Using IQFEED" on page 1-30

Introduced in R2012b

close

Close IQFEED ports

Syntax

`close(Q)`

Description

`close(Q)` closes all IQFEED ports currently open for a given IQFEED connection handle, `Q`.

Arguments

`Q`	IQFEED connection handle created using `iqf`.

Examples

Close all ports for an IQFEED connection handle.

```
close(Q)
```

See Also

See Also
`iqf`

Topics
"Retrieve Intraday and Historical Data Using IQFEED" on page 1-30

Introduced in R2012b

history

IQFEED asynchronous historical end-of-period data

Syntax

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

history(c,s,{startdate,enddate})
history(c,s,{startdate,enddate},[],listener,eventhandler)
```

Description

`history(c,s,interval)` returns asynchronous historical end-of-period data using the connection object `c`, a single security `s`, and a specified interval `interval`.

`history(c,s,interval,period)` returns asynchronous historical end-of-period data for a single security with a specified interval and period `period`.

`history(c,s,interval,period,listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a specified interval, period, socket listener `listener`, and event handler `eventhandler`.

`history(c,s,{startdate,enddate})` returns asynchronous historical end-of-period data for a single security with a date range.

`history(c,s,{startdate,enddate},[],listener,eventhandler)` returns asynchronous historical end-of-period data for a single security with a date range, a specified socket listener `listener`, and event handler `eventhandler`.

Examples

Retrieve Daily Data

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five days.

```
history(c,'GOOG',5)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:08:58'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:08:58'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:08:58'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
    '2013-11-18 11:08:58'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
    '2013-11-15 11:08:58'    '1038.00'    '1030.31'    '1034.87'    '1033.56'    '1277772'    '0'
```

Each row of data represents one day. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Weekly Data

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five weeks.

```
history(c,'GOOG',5,'Weekly')
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:07:02'    '1048.74'    '1020.36'    '1035.75'    '1034.07'    '4949900'     '0'
    '2013-11-15 11:07:02'    '1039.75'    '1005.00'    '1009.51'    '1033.56'    '6361983'     '0'
    '2013-11-08 11:07:02'    '1032.37'    '1007.64'    '1031.50'    '1016.03'    '6209876'     '0'
    '2013-11-01 11:07:02'    '1041.52'    '1012.98'    '1015.20'    '1027.04'    '7025769'     '0'
    '2013-10-25 11:07:02'    '1040.57'    '995.79'     '1011.46'    '1015.20'    '12636223'    '0'
```

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

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Monthly Data with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five months. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data.

```
history(c,'GOOG',5,'Monthly',@iqhistoryfeedlistener,...
        @iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:13:07'    '1048.74'    '1005.00'    '1031.79'    '1034.07'    '18805697'    '0'
    '2013-10-31 11:13:07'    '1041.52'    '842.98'     '880.25'     '1030.58'    '55288774'    '0'
    '2013-09-30 11:13:07'    '905.99'     '853.95'     '854.36'     '875.91'     '33147210'    '0'
    '2013-08-30 11:13:07'    '909.71'     '845.56'     '895.00'     '846.90'     '33509358'    '0'
    '2013-07-31 11:13:07'    '928.00'     '875.61'     '886.45'     '887.75'     '51277966'    '0'
```

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

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Data for a Date Range

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve IBM security data for the last five days.

```
history(c,'IBM',{floor(now-5),floor(now)})
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =
```

```
    '2013-11-21 10:59:51'    '185.7500'    '183.4110'    '185.5400'    '184.1300'    '4459451'    '0'
    '2013-11-20 10:59:51'    '186.2400'    '184.6450'    '185.2200'    '185.1900'    '3646117'    '0'
    '2013-11-19 10:59:51'    '186.2000'    '184.1500'    '184.6300'    '185.2500'    '4577037'    '0'
    '2013-11-18 10:59:51'    '184.9900'    '183.2700'    '183.5200'    '184.4700'    '5344864'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

Retrieve Data for a Date Range with Event Handlers

Create the IQFEED connection with user name `username` and password `pwd`.

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

Retrieve the Google security data for the last five days. Use the event handler functions `iqhistoryfeedlistener` and `iqhistoryfeedeventhandler` to listen for the Google security and parse the resulting data. The period `[]` specifies the default period for daily data.

```
history(c,'GOOG',{floor(now-5),floor(now)},[],...
        @iqhistoryfeedlistener,@iqhistoryfeedeventhandler)
```

`history` returns the data in the MATLAB cell array `IQFeedHistoryData`.

Display the returned data in `IQFeedHistoryData`.

```
IQFeedHistoryData
```

```
IQFeedHistoryData =

    '2013-11-21 11:12:15'    '1038.31'    '1026.00'    '1027.00'    '1034.07'    '1092497'    '0'
    '2013-11-20 11:12:15'    '1033.36'    '1020.36'    '1029.95'    '1022.31'    '965535'     '0'
    '2013-11-19 11:12:15'    '1034.75'    '1023.05'    '1031.72'    '1025.20'    '1131619'    '0'
```

```
'2013-11-18 11:12:15'    '1048.74'    '1029.24'    '1035.75'    '1031.55'    '1760249'    '0'
```

Each row of data represents one day. Since this example is run on a Friday, the return data has only four days. The columns in `IQFeedHistoryData` contain the following:

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

Close the IQFEED connection.

```
close(c)
```

- "Retrieve Intraday and Historical Data Using IQFEED" on page 1-30

Input Arguments

`c` — IQFEED connection

connection object

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

`s` — Security

character vector

Security, specified as a character vector 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 preceding values to denote daily, weekly, or monthly return data. When this argument is specified along with `interval`, `history` returns the number of daily, weekly, or monthly data where the number of output rows corresponds to the `interval`. When this argument is omitted by specifying `[]`, `history` returns daily data.

`listener` — Listener event handler

function

Listener event handler, specified as a function to listen for the IQFEED data. You can modify the existing listener function or define your own. You can find the code for the existing listener function in the `history.m` file.

Data Types: `function_handle`

`eventhandler` — Event handler

function

Event handler, specified as a function to process the IQFEED data. The existing event handler displays the IQFEED data in the Command Window. You can modify the existing event handler function or 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

numeric scalar | character vector

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

Example: `floor(now-1)`

Data Types: `double` | `char`

`enddate` — End date

numeric scalar | character vector

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

Example: `floor(now)`

Data Types: `double` | `char`

Tips

- When you make multiple requests with multiple messages, this error might occur:

  Warning: Error occurred while executing delegate callback: Message: The IAsyncResult object was not returned from the corresponding asynchronous method on this class.

  To fix this, restart MATLAB.

See Also

See Also

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

Topics

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-30
“Writing and Running Custom Event Handler Functions” on page 1-35

Introduced in R2012b

marketdepth

IQFEED asynchronous level 2 data

Syntax

```
marketdepth(Q,S)
marketdepth(Q,S,elistener,ecallback)
```

Description

`marketdepth(Q,S)` returns asynchronous level 2 data using the default socket listener and event handler.

`marketdepth(Q,S,elistener,ecallback)` returns asynchronous level 2 data using an explicitly defined socket listener and event handler.

Arguments

`Q`	IQFEED connection handle created using `iqf`.
`S`	`S` is specified as a character vector for a single security or a cell array of character vectors for multiple securities.
`elistener`	Function handle that specifies the function used to listen for data on the level 2 port.
`ecallback`	Function handle that specifies the function that processes data event.

Examples

Return level 2 data using the default socket listener and event handler and display the results in the MATLAB workspace in the variable `IQFeedLevelTwoData`.

```
marketdepth(q,'ABC')
```

```
openvar('IQFeedLevelTwoData')
```

Initiate a watch on the security ABC for level 2 data using the function handles `iqfeedlistener` and `iqfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedLevelTwoData`.

```
marketdepth(q,'ABC',@iqfeedmarketdepthlistener,@iqfeedmarketdeptheventhandler)
openvar('IQFeedLevelTwoData')
```

See Also

See Also

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

Topics

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-30
“Writing and Running Custom Event Handler Functions” on page 1-35

Introduced in R2012b

news

IQFEED asynchronous news data

Syntax

```
news(Q,S)
news(Q,S,elistener,ecallback)
```

Description

`news(Q,S)` returns asynchronous news data using the default socket listener and event handler.

`news(Q,S,elistener,ecallback)` returns asynchronous news data using an explicitly defined socket listener and event handler.

The syntax `news(Q,true)` turns on news updates for the list of currently subscribed level 1 securities and `news(Q,false)` turns off news updates for the list of currently subscribed level 1 securities.

Arguments

`Q`	IQFEED connection handle created using `iqf`.
`S`	`S` is specified as a character vector for a single security or a cell array of character vectors for multiple securities.
`elistener`	Function handle that specifies the function used to listen for data on the news lookup port.
`ecallback`	Function handle that specifies the function that processes data events.

Examples

Return news data using the defaults for socket listener and event handler and display the results in the MATLAB workspace in the variable `IQFeedNewsData`.

```
news(q,'ABC')
openvar('IQFeedNewsData')
```

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

```
news(q,'ABC',@iqfeednewslistener,@iqfeednewseventhandler)
openvar('IQFeedNewsData')
```

See Also

See Also

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

Topics

"Retrieve Intraday and Historical Data Using IQFEED" on page 1-30
"Writing and Running Custom Event Handler Functions" on page 1-35

Introduced in R2012b

realtime

IQFEED asynchronous level 1 data

Syntax

```
realtime(Q,S)
realtime(Q,S,F)
realtime(Q,S,elistener,ecallback)
```

Description

`realtime(Q,S)` returns asynchronous level 1 data using the current update field list, default socket listener, and event handler.

`realtime(Q,S,F)` returns asynchronous level 1 data for a specified field list using the default socket listener and event handler.

`realtime(Q,S,elistener,ecallback)` returns asynchronous level 1 data using an explicitly defined socket listener and event handler.

Arguments

`Q`	IQFEED connection handle created using `iqf`.
`S`	`S` is specified as a character vector for a single security or a cell array of character vectors for multiple securities.
`F`	`F` is the field list. If no field list is specified or it is input as empty, the default IQFEED level 1 field will be updated with each tick.
`elistener`	Function handle that specifies the function used to listen for data on the IQFEED Lookup port.
`ecallback`	Function handle that specifies the function that processes data event.

Examples

Set the data precision. Setting the connection handle property `Protocol` determines the date format for the return data based on the IQFEED version specified by the protocol.

```
q.Protocol = 5.1

q = 

  iqf with properties:

        User: 'username'
    Password: 'password'
        Port: {[1x1 System.Net.Sockets.Socket]}
    PortName: {'Admin'}
    Protocol: 5.1000
```

Return level 1 data for security ABC using the default socket listener and event handler. Display the results in the MATLAB workspace in the variable `IQFeedLevelOneData`.

```
realtime(q,'ABC')
openvar('IQFeedLevelOneData')
```

Return level 1 data for security ABC using a field list and the defaults for the socket listener and event handler. Display the results in the MATLAB workspace in the variable `IQFeedLevelOneData`.

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

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

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

See Also

See Also

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

Topics

"Retrieve Intraday and Historical Data Using IQFEED" on page 1-30
"Writing and Running Custom Event Handler Functions" on page 1-35

Introduced in R2012b

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 character vector.
`daterange`	Either a scalar value that specifies how many periods of data to return or a date range of the form`{startdate,enddate}`. `startdate` and `enddate` can be input as MATLAB date numbers or character vectors.
`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. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{floor(now),now})
openvar('IQFeedTimeseriesData')
```

For data that is not aggregated, the fields returned are:

- Timestamp
- Last
- Last size
- Total volume
- Bid
- Ask
- TickID
- IQFEED reserved field
- IQFEED reserved field
- Basis for last

Basis for last is either a `C` that means the last qualified trade or `E` that means an extended trade.

Return the intraday ticks for a date range using the 24-hour military format, `per` of 60 seconds, and the default socket listener and event handler. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

```
timeseries(q,'ABC',{'02/12/2012 09:30:00','02/12/2012 16:00:00'},60)
openvar('IQFeedTimeseriesData')
```

For aggregated data, the fields returned are:

- Timestamp

- High
- Low
- Open
- Close
- Total volume
- Period volume

Return the intraday ticks for a date range using the 12-hour time format.

```
timeseries(q,'ABC',{'02/12/2012 09:30:00 AM','02/12/2012 04:00:00 PM'},60)
openvar('IQFeedTimeseriesData')
```

Return the intraday ticks for a date range on the security `ABC` using the function handles `iqtimeseriesfeedlistener` and `iqtimeseriesfeedeventhandler`. Display the results in the MATLAB workspace in the variable `IQFeedTimeseriesData`.

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

Tips

- When you make multiple requests with multiple messages, this error might display: Warning: Error occurred while executing delegate callback: Message: The IAsyncResult object was not returned from the corresponding asynchronous method on this class.

  To fix this, restart MATLAB.

See Also

See Also

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

Topics

“Retrieve Intraday and Historical Data Using IQFEED” on page 1-30
“Writing and Running Custom Event Handler Functions” on page 1-35

Introduced in R2012b

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

See Also

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

Introduced before R2006a

close

Close connection to FactSet

Syntax

```
close(Connect)
```

Arguments

`Connect`	FactSet connection object created with `factset`.

Description

`close(Connect)` closes the connection to FactSet data.

See Also

See Also

`factset`

Introduced before R2006a

fetch

Request data from FactSet

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`Library`	FactSet formula library.
`Security`	A MATLAB character vector or cell array of character vectors containing the names of securities in a format recognizable by the FactSet server.
`Fields`	A MATLAB character vector or cell array of character vectors indicating the data fields for which to retrieve data.
`Date`	Date character vector 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

See Also

`close` | `factset` | `isconnection`

Introduced before R2006a

get

Retrieve properties of FactSet connection object

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`PropertyName`	(Optional) A MATLAB character vector or cell array of character vectors 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

See Also

`close` | `factset` | `fetch` | `isconnection`

Introduced before R2006a

isconnection

Determine if connections to FactSet are valid

Syntax

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

Arguments

`Connect`	FactSet connection object created with `factset`.

Description

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

Examples

Establish a connection, `c`, to FactSet data:

```
c = factset
```

Verify that `c` is a valid connection:

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

See Also

See Also

`close` | `factset` | `fetch` | `get`

Introduced before R2006a

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

character vector

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

Data Types: `char`

`Password` — User password

character vector

User password to FactSet Data Server, specified as a character vector.

Data Types: `char`

`Finfo` — Field information

character vector

Field information, specified as a character vector.

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

See Also

`close` | `realtime` | `stop`

Introduced in R2013a

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

character vector

Data source or supplier, specified as a character vector.

Example: `'FDS1'`

Data Types: `char`

`Sec` — Security symbol

character vector

Security symbol, specified as a character vector.

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.

See Also

See Also
`close` | `fds` | `stop`

Topics
"Writing and Running Custom Event Handler Functions" on page 1-35

Introduced in R2013a

stop

Cancel real-time request

Syntax

`stop(c,T)`

Description

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

Examples

Cancel FactSet 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

See Also
`close` | `fds` | `realtime`

Introduced in R2013a

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

See Also

`fds` | `realtime` | `stop`

Introduced in R2013a

fred

Connect to FRED data servers

Syntax

```
c = fred
c = fred(url)
```

Description

`c = fred` returns a FRED connection to the FRED data server using the default URL `'https://research.stlouisfed.org/fred2/'`.

`c = fred(url)` returns a FRED connection using a specific URL.

Examples

Connect to FRED®

Connect to the FRED® data server.

```
c = fred
```

```
c = 

  fred with properties:

     url: 'https://research.stlouisfed.org/fred2/'
      ip: []
    port: []
```

`c` is a FRED® connection with these properties:

- url -- The URL for the FRED® data server
- ip -- The IP address of the proxy server
- port -- The port number of the proxy server

Retrieve the `ip` property of the FRED® connection `c`.

```
c.ip
```

```
ans =

     []
```

Retrieve the `port` property of the FRED® connection `c`.

```
c.port
```

```
ans =

     []
```

Close the FRED® connection.

```
close(c)
```

Connect to FRED® with a URL

Connect to the FRED® data server using the URL https://research.stlouisfed.org/fred2/.

```
url = 'https://research.stlouisfed.org/fred2/';
c = fred(url)
```

```
c =

  fred with properties:

      url: 'https://research.stlouisfed.org/fred2/'
       ip: []
     port: []
```

`c` is a FRED® connection with these properties:

- url -- The URL for the FRED® data server
- ip -- The IP address of the proxy server

- port -- The port number of the proxy server

Retrieve the `ip` property of the FRED® connection `c`.

```
c.ip

ans =

     []
```

Retrieve the `port` property of the FRED® connection `c`.

```
c.port

ans =

     []
```

Close the FRED® connection.

```
close(c)
```

- “Retrieve Historical Data Using FRED®” on page 1-25

Input Arguments

`url` — URL of FRED data server
character vector

URL of the FRED data server, specified as a character vector.

Example: `'https://research.stlouisfed.org/fred2/'`

Data Types: `char`

Output Arguments

`c` — FRED connection
connection object

FRED connection, returned as a connection object with these properties.

Property	Description
`url`	URL of the FRED data server
`ip`	IP address of the proxy server
`port`	Port number of the proxy server

See Also

See Also

`close` | `fetch` | `isconnection`

Topics

"Retrieve Historical Data Using FRED®" on page 1-25

Introduced in R2006b

close

Close connections to FRED data servers

Syntax

```
close(c)
```

Arguments

`c`	FRED connection object created with `fred`.

Description

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

Examples

Close FRED® Connection

Connect to the data server at the URL https://research.stlouisfed.org/fred2/.

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

Adjust the display data format for currency.

```
format bank
```

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

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

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

`d` contains the series description.

Close the FRED® connection.

```
close(c);
```

- “Retrieve Historical Data Using FRED®” on page 1-25

See Also

See Also

`fred`

Topics

“Retrieve Historical Data Using FRED®” on page 1-25

Introduced in R2006b

fetch

Request data from FRED data servers

Syntax

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

Description

`d = fetch(c,series)` returns FRED data using the FRED connection `c` and the specified FRED series.

`d = fetch(c,series,date)` returns FRED data for a specific date.

`d = fetch(c,series,startdate,enddate)` returns FRED data for the date range from `startdate` through `enddate`.

Examples

Fetch All Available FRED® Data

Connect to the FRED® data server using the URL `'https://research.stlouisfed.org/fred2/'`.

```
url = 'https://research.stlouisfed.org/fred2/';
c = fred(url);
```

Fetch all available daily foreign exchange rates between the US dollar and the Euro using the series `'DEXUSEU'`.

```
series = 'DEXUSEU';
d = fetch(c,series)
```

```
d =
```

```
  struct with fields:

                 Title: ' U.S. / Euro Foreign Exchange Rate'
              SeriesID: ' DEXUSEU'
                Source: ' Board of Governors of the Federal Reserve System...'
               Release: ' H.10 Foreign Exchange Rates'
    SeasonalAdjustment: ' Not Seasonally Adjusted'
             Frequency: ' Daily'
                 Units: ' U.S. Dollars to One Euro'
             DateRange: ' 1999-01-04 to 2016-07-08'
           LastUpdated: ' 2016-07-11 3:41 PM CDT'
                 Notes: ' Noon buying rates in New York City for cable tra...'
                  Data: [4570×2 double]
```

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

Close the FRED® connection.

```
close(c)
```

Fetch FRED® Data for a Date

Connect to the FRED® data server using the URL `'https://research.stlouisfed.org/fred2/'`.

```
url = 'https://research.stlouisfed.org/fred2/';
c = fred(url);
```

Fetch data for a day three months ago using the series `'DTB6'`.

```
series = 'DTB6';
date = floor(now)-90;
d = fetch(c,series,date)
```

```
d =

  struct with fields:

                 Title: ' 6-Month Treasury Bill: Secondary Market Rate'
              SeriesID: ' DTB6'
                Source: ' Board of Governors of the Federal Reserve System...'
```

```
              Release: ' H.15 Selected Interest Rates'
   SeasonalAdjustment: ' Not Seasonally Adjusted'
            Frequency: ' Daily'
                Units: ' Percent'
            DateRange: ' 1958-12-09 to 2016-07-11'
          LastUpdated: ' 2016-07-12 3:46 PM CDT'
                Notes: ' Discount Basis'
                 Data: [736434 0.3600]
```

`d.Data` is an N-by-2 double array that contains the date in the first column and the series value in the second column.

Close the FRED® connection.

```
close(c)
```

Fetch FRED® Data for a Date Range

Connect to the FRED® data server using the URL `'https://research.stlouisfed.org/fred2/'`.

```
url = 'https://research.stlouisfed.org/fred2/';
c = fred(url);
```

Fetch five months of data from January 1, 2007 through June 1, 2007.

```
series = 'DEXUSEU';
startdate = '01/01/2007';
enddate = '06/01/2007';
d = fetch(c,series,startdate,enddate)
```

```
d =

  struct with fields:

                Title: ' U.S. / Euro Foreign Exchange Rate'
             SeriesID: ' DEXUSEU'
               Source: ' Board of Governors of the Federal Reserve System...'
              Release: ' H.10 Foreign Exchange Rates'
   SeasonalAdjustment: ' Not Seasonally Adjusted'
            Frequency: ' Daily'
                Units: ' U.S. Dollars to One Euro'
            DateRange: ' 1999-01-04 to 2016-07-08'
```

```
    LastUpdated: ' 2016-07-11 3:41 PM CDT'
          Notes: ' Noon buying rates in New York City for cable tra...'
           Data: [110×2 double]
```

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

Close the FRED® connection.

```
close(c)
```

- “Retrieve Historical Data Using FRED®” on page 1-25

Input Arguments

`c` — FRED connection
connection object

FRED connection, specified as a connection object created using `fred`.

`series` — FRED series
character vector

FRED series, specified as a character vector.

Example: `'DEXUSEU'`

Data Types: `char`

`date` — Date
`datetime` | matrix | character vector

Date, specified as a `datetime` value, matrix, or date character vector. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime`

`startdate` — Start date
`datetime` | matrix | character vector

Start date in a date range, specified as a `datetime` value, matrix, or date character vector. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime`

`enddate` — End date
`datetime` | matrix | character vector

End date in a date range, specified as a `datetime` value, matrix, or date character vector. For details about the data types, see `datenum`.

Data Types: `double` | `char` | `cell` | `datetime`

Output Arguments

`d` — FRED data
structure

FRED data, returned as a structure. For details about FRED data, see https://research.stlouisfed.org/fred2/.

See Also

See Also
`close` | `fred` | `isconnection`

Topics
"Retrieve Historical Data Using FRED®" on page 1-25

Introduced in R2006b

get

Retrieve properties of FRED connection objects

Syntax

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

Arguments

`c`	FRED connection object created with `fred`.
`'PropertyName'`	A MATLAB character vector or cell array of character vectors containing property names. Property names are: • `'url'` • `'ip'` • `'port'`

Description

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

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

Examples

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

```
c = fred('https://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

See Also

close | fetch | isconnection

Topics

"Retrieve Historical Data Using FRED®" on page 1-25

Introduced in R2006b

isconnection

Determine if connections to FRED data servers are valid

Syntax

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

Arguments

`c`	FRED connection object created with `fred`.

Description

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

Examples

Verify FRED® Connection

Establish a connection `c` to a FRED® data server.

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

Verify that `c` is a valid connection.

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

```
x =

     1
```

Adjust the display data format for currency.

```
format bank
```

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

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

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

`d` contains the series description.

Close the FRED® connection.

```
close(c)
```

- “Retrieve Historical Data Using FRED®” on page 1-25

See Also

See Also

`close` | `fetch` | `fred`

Topics

“Retrieve Historical Data Using FRED®” on page 1-25

Introduced in R2006b

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

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

aggregation

Set Haver Analytics aggregation mode

Syntax

```
X = aggregation (C)
X = aggregation (C,V)
```

Description

`X = aggregation (C)` returns the current aggregation mode.

`X = aggregation (C,V)` sets the current aggregation mode to `V`. The following table lists possible values for `V`.

Value of `V`	Aggregation mode	Behavior of `aggregation` function
`0`	`strict`	`aggregation` does not fill in values for missing data.
`1`	`relaxed`	`aggregation` fills in missing data based on data available in the requested period.
`2`	`forced`	`aggregation` fills in missing data based on some past value.
`-1`	Not recognized	`aggregation` resets `V` to its last valid setting.

See Also

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2008b

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

See Also

`haver`

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

fetch

Request data from Haver Analytics database

Syntax

```
d = fetch(c,variable)
d = fetch(c,variable,startdate,enddate)
d = fetch(c,variable,startdate,enddate,period)
```

Description

`d = fetch(c,variable)` returns historical data for the Haver Analytics variable `s`, using the connection object `c`.

`d = fetch(c,variable,startdate,enddate)` returns historical data between the dates `startdate` and `enddate`.

`d = fetch(c,variable,startdate,enddate,period)` returns historical data in time periods specified by `period`.

Examples

Retrieve Variable Data

Connect to the Haver Analytics database.

```
c = haver('c:\work\haver\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)
```

- "Retrieve Historical Data Using Haver Analytics" on page 1-28

Input Arguments

c — Haver Analytics connection

connection object

Haver Analytics connection, specified as a connection object created using `haver`.

variable — Haver Analytics variable

character vector

Haver Analytics variable, specified as a character vector to denote which historical data to retrieve.

Example: `'FFED'`

Data Types: `char`

startdate — Start date

character vector | MATLAB date number

Start date, specified as a character vector or MATLAB date number denoting the beginning of the date range to retrieve data.

Data Types: `double` | `char`

enddate — End date

character vector | MATLAB date number

End date, specified as a character vector 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 these values that denotes the time period for the historical data:

- `'d'` for daily values
- `'w'` for weekly values
- `'m'` for monthly values
- `'q'` for quarterly values

- `'a'` for annual values

Output Arguments

`d` — Historical data
matrix

Historical data, returned as a matrix with the numeric representation of the date in the first column and the value in the second column.

See Also

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

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 character vector or cell array of character vectors 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

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

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

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

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

See Also

close | fetch | get | haver

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

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

See Also

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

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

havertool

Run Haver Analytics graphical user interface (GUI)

Syntax

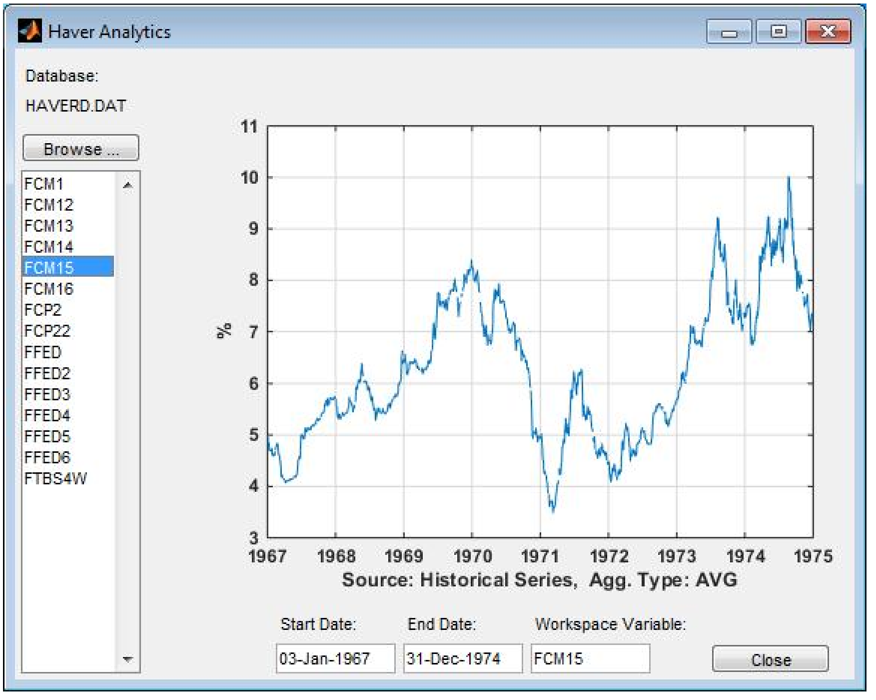
`havertool(H)`

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

See Also

`haver`

Topics

"Retrieve Historical Data Using Haver Analytics" on page 1-28

Introduced in R2007a

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* character vector 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* character vector `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

See Also

close | exec | fetch | get | tables

Introduced in R2007a

close

Close connections to Kx Systems, Inc. kdb+ databases

Syntax

```
close(k)
```

Arguments

k	Kx Systems, Inc. kdb+ connection object created with `kx`.

Description

`close(k)` closes the connection to the Kx Systems, Inc. kdb+ database.

Examples

Close the connection, `k`, to the Kx Systems, Inc. kdb+ database:

```
close(k)
```

See Also

See Also

`kx`

Introduced in R2007a

exec

Run Kx Systems, Inc. kdb+ commands

Syntax

```
exec(k,command)
exec(k,command,p1,p2,p3)
exec(k,command,p1)
exec(k,command,p1,p2)
exec(k,command,p1,p2,p3)
exec(k,command,p1,p2,p3,sync)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with `kx`.
`command`	Kx Systems, Inc. kdb+ command issued using the Kx Systems, Inc. kdb + connection object created with the `kx` function.
`p1,p2,p3`	Input parameters for `Command`.

Description

`exec(k,command)` executes the specified `command` in Kx Systems, Inc. kdb+ without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the specified `command` with one or more input parameters without waiting for a response.

`exec(k,command,p1)` executes the given `command` with one input parameter without waiting for a response.

`exec(k,command,p1,p2)` executes the given `command` with two input parameters without waiting for a response.

`exec(k,command,p1,p2,p3)` executes the given `command` with three input parameters without waiting for a response.

`exec(k,command,p1,p2,p3,sync)` executes the given `command` with three input parameters synchronously and waits for a response from the database. Enter unused parameters as empty. You can enter `sync` as 0 (default) for asynchronous commands and as 1 for synchronous commands.

Examples

Retrieve the data in the table `trade` using the connection to the Kx Systems, Inc. kdb+ database, `K`:

```
k = kx('localhost',5001);
```

Use the `exec` command to sort the data in the table `trade` in ascending order.

```
exec(k,'`date xasc`trade');
```

Subsequent data requests also sort returned data in ascending order.

After running

```
q tradedata.q -p 5001
```

at the DOS prompt, the commands

```
k = kx('localhost',5001);
exec(k,'`DATE XASC `TRADE');
```

sort the data in the table `trade` in ascending order. Data later fetched from the table will be ordered in this manner.

See Also

See Also

`fetch` | `insert` | `kx`

Introduced in R2007a

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

See Also

`exec` | `insert` | `kx`

Introduced in R2007a

get

Retrieve Kx Systems, Inc. kdb+ connection object properties

Syntax

```
v = get(k,'PropertyName')
v = get(k)
```

Arguments

k	Kx Systems, Inc. kdb+ connection object created with kx.
'PropertyName'	A character vector or cell array of character vectors 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

See Also
`close` | `exec` | `fetch` | `insert` | `kx`

Introduced in R2007a

insert

Write data to Kx Systems, Inc. kdb+ databases

Syntax

```
insert(k,tablename,data)
x = insert(k,tablename,data,sync)
```

Arguments

k	The Kx Systems, Inc. kdb+ connection object created with kx.
tablename	The name of the Kx Systems, Inc. kdb+ Tablename.
data	The data that insert writes to the Kx Systems, Inc. kdb+ Tablename.

Description

insert(k,tablename,data) writes the data, data, to the Kx Systems, Inc. kdb+ table, tablename.

x = insert(k,tablename,data,sync) writes the data, data, to the Kx Systems, Inc. kdb+ table, tablename, synchronously. For asynchronous calls, enter sync as 0 (default), and for synchronous calls, enter sync as 1.

Examples

For the connection to the Kx Systems, Inc. kdb+ database, k, write data from ACME to the specified table:

```
insert(k,'trade',{'`ACME',133.51,250,6.4,'2006.10.24'})
```

See Also

See Also

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

Introduced in R2007a

isconnection

Determine if connections to Kx Systems, Inc. kdb+ databases are valid

Syntax

```
x = isconnection(k)
```

Arguments

k	Kx Systems, Inc. kdb+ connection object created with kx.

Description

`x = isconnection(k)` returns `x = 1` if the connection to the Kx Systems, Inc. kdb+ database is valid, and `x = 0` otherwise.

Examples

Establish a connection to a Kx Systems, Inc. kdb+ database, `k`:

```
k = kx('localhost',5001);
```

Verify that `k` is a valid connection:

```
x = isconnection(k)
x = 1
```

See Also

See Also

`close` | `fetch` | `get` | `kx`

Introduced in R2007a

tables

Retrieve table names from Kx Systems, Inc. kdb+ databases

Syntax

`t = tables(k)`

Arguments

`k`	The Kx Systems, Inc. kdb+ connection object created with the `kx` function.

Description

`t = tables(k)` returns the list of tables for the Kx Systems, Inc. kdb+ connection.

Examples

Retrieve table information for the Kx Systems, Inc. kdb+ database using the connection `k`:

```
t = tables(k)
t =
     'intraday'
   'seclist'
     'trade'
```

See Also

See Also

`exec` | `fetch` | `insert` | `kx`

Introduced in R2007a

moneynet

Create Money.Net connection

Description

The `moneynet` function creates a `moneynet` object. The `moneynet` object represents a Money.Net connection.

You can retrieve current, intraday, historical, real-time, and news data using the `moneynet` object. You can retrieve data based on your credentials that consist of a user name and password. For details about credentials, contact Money.Net.

Create Object

Syntax

Description

`c = moneynet(username,password)` creates a Money.Net connection `c` using the user name and password.

`c = moneynet(username,password,portnumber)` specifies a port number for the connection.

Input Arguments

`username` — User name
character vector | string scalar

User name required to access Money.Net data, specified as a character vector or string scalar. The user name is an email address. To request your Money.Net user name, contact Money.Net.

Data Types: `char`

`password` — Password
character vector | string scalar

Password required to access Money.Net data, specified as a character vector or string scalar. To request your Money.Net password, contact Money.Net.

Data Types: `char`

`portnumber` — Port number
`50010` (default) | numeric scalar

Port number of the Money.Net data server, specified as a numeric scalar.

Data Types: `double`

Properties

`Username` — User name
character vector | string scalar

User name, specified as a character vector or string scalar.

Data Types: `char`

`Port` — Port number
`50010` (default) | numeric scalar

Port number of the Money.Net data server, specified as a numeric scalar.

Data Types: `double`

`Server` — Money.Net server name
character vector

Money.Net server name, specified as a character vector.

Data Types: `char`

Object Functions

close	Close Money.Net connection
getdata	Retrieve Money.Net current data

isconnection	Determine if Money.Net connection is valid
timeseries	Retrieve Money.Net intraday and historical data
realtime	Retrieve Money.Net real-time data
stop	Unsubscribe Money.Net real-time data updates
getsubscriptions	Retrieve Money.Net subscribed symbols and event handler functions
optionchain	Retrieve Money.Net option symbols
news	Search and stream Money.Net latest news stories

Examples

Connect to Money.Net

Connect to Money.Net using a user name and password.

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

c = moneynet(username,pwd)

c = 

  moneynet with properties:

    Username: 'user@company.com'
        Port: 50010
      Server: 'NTY_JAMES_IRWIN_88 TCP'
```

`c` is the Money.Net connection object.

Verify the Money.Net connection `c` using `isconnection`.

```
v = isconnection(c)

v = 

  logical

   1
```

`isconnection` returns `1`, indicating a successful connection.

Close the Money.Net connection.

```
close(c)
```

Connect to Money.Net Using Port Number

Connect to Money.Net using a user name, password, and port number.

```
username = 'user@company.com';
pwd = '999999';
portnumber = 50010;

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

```
c =

  moneynet with properties:

    Username: 'user@company.com'
        Port: 50010
      Server: 'NTY_JAMES_IRWIN_88 TCP'
```

`c` is the Money.Net connection object.

Verify the Money.Net connection `c` using `isconnection`.

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

```
v =

  logical

   1
```

`isconnection` returns `1`, indicating a successful connection.

Close the Money.Net connection.

```
close(c)
```


- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-6
- “Retrieve Money.Net News Stories” on page 5-9

See Also

Topics

“Retrieve Current and Historical Money.Net Data” on page 5-2
“Retrieve Real-Time Money.Net Data” on page 5-6
“Retrieve Money.Net News Stories” on page 5-9
“Money.Net Error and Warning Messages” on page 5-13

External Websites

Money.Net API Documentation
Money.Net Help

Introduced in R2016b

close

Close Money.Net connection

Syntax

```
close(c)
```

Description

`close(c)` closes the Money.Net connection `c`.

Examples

Close Money.Net Connection

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

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

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

Close the Money.Net connection.

```
close(c)
```

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-6
- “Retrieve Money.Net News Stories” on page 5-9

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

See Also

See Also
`getdata` | `isconnection` | `moneynet` | `news` | `realtime` | `timeseries`

Topics
“Retrieve Current and Historical Money.Net Data” on page 5-2
“Retrieve Real-Time Money.Net Data” on page 5-6
“Retrieve Money.Net News Stories” on page 5-9
“Money.Net Error and Warning Messages” on page 5-13

External Websites
Money.Net API Documentation

Introduced in R2016b

isconnection

Determine if Money.Net connection is valid

Syntax

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

Description

`v = isconnection(c)` returns logical `1 (true)` if `c` is a valid Money.Net connection. Otherwise, `isconnection` returns logical `0 (false)`.

Examples

Validate Money.Net Connection

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

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

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

Validate the Money.Net connection `c`.

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

```
v =

  logical

   1
```

`isconnection` returns `1`, indicating a successful connection.

Close the Money.Net connection.

```
close(c)
```

Validate that the Money.Net connection `c` is closed.

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

```
v =

  logical

   0
```

`isconnection` returns `0`, indicating a closed connection.

- “Retrieve Current and Historical Money.Net Data” on page 5-2
- “Retrieve Real-Time Money.Net Data” on page 5-6
- “Retrieve Money.Net News Stories” on page 5-9

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

Output Arguments

v — Valid Money.Net connection

`true` | `false`

Valid Money.Net connection, returned as a logical `1 (true)` that specifies a successful connection, or logical `0 (false)` that specifies a closed or invalid connection.

See Also

See Also

`close` | `moneynet`

Topics

"Retrieve Current and Historical Money.Net Data" on page 5-2
"Retrieve Real-Time Money.Net Data" on page 5-6
"Retrieve Money.Net News Stories" on page 5-9
"Money.Net Error and Warning Messages" on page 5-13

External Websites

Money.Net API Documentation

Introduced in R2016b

getdata

Retrieve Money.Net current data

Syntax

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

Description

`d = getdata(c,symbols,f)` returns Money.Net data `d` using the Money.Net connection `c` for the symbols and the Money.Net fields `f`.

Examples

Retrieve Current Data for One Symbol

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

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

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

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

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

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

Display Money.Net current data.

```
d
```

```
d =
```

```
    Symbol     Ask        Bid  
    ______    ______    ______

    'IBM'     145.00    143.85
```

d is a table that contains the columns for symbol, ask price, and bid price. The row contains the Money.Net data values for each column.

Close the Money.Net connection.

```
close(c)
```

Retrieve Current Data for Multiple Symbols

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

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

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

Retrieve Money.Net current data d for the symbols list that contains IBM, Google, and Yahoo! using the Money.Net connection c. Specify the Money.Net data fields f for ask and bid price.

```
symbols = {'IBM','GOOG','YHOO'};
f = {'Ask','Bid'};

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

Display Money.Net current data.

```
d

d = 

    Symbol     Ask        Bid  
    ______    ______    ______

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

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

Close the Money.Net connection.

```
close(c)
```

- “Retrieve Current and Historical Money.Net Data” on page 5-2

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

symbols — Money.Net symbol list

character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char`

f — Money.Net data field list

character vector | cell array of character vectors | string scalar | string array

Money.Net data field list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one field, use a character vector or string scalar. To specify multiple fields, use a cell array of character vectors or a string array.

Specify the field by using the single character or the field definition. For example, to specify the highest price for the equity during the current trading day, use a single character `'H'` or the corresponding field definition `'High'`. When using the field definition, the software ignores the case of the definition. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation.

Example: `'High'`

Example: `{'High','Low'}`

Data Types: `char`

Output Arguments

d — Money.Net data

table

Money.Net data, returned as a table. Each row corresponds to the `symbols` list. Each column corresponds to the field list `f`.

See Also

See Also

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

Topics

"Retrieve Current and Historical Money.Net Data" on page 5-2
"Money.Net Error and Warning Messages" on page 5-13

External Websites

Money.Net API Documentation

Introduced in R2016b

timeseries

Retrieve Money.Net intraday and historical data

Syntax

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

Description

`d = timeseries(c,s,date,interval)` returns Money.Net intraday and historical data using the Money.Net connection `c` for all available fields. Specify the Money.Net symbol `s` and the current or historical date. To specify the amount of data to return, use the bar interval.

`d = timeseries(c,s,date,interval,f)` returns Money.Net intraday and historical data for the specified Money.Net fields `f`.

Examples

Retrieve Intraday Data in Seconds

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

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

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

Retrieve intraday data for the last 5 minutes in 30-second bars for the symbol IBM using the Money.Net connection `c`. Specify the date as a `datetime` array containing a date range with start and end dates. The start date starts 5 minutes after the current moment. The end date is the current moment. To specify the current moment, use `datetime('now')`. To specify 5 minutes earlier, subtract `minutes(5)` from the current moment. To retrieve data in 30-second bars, specify the interval as `'30S'`.

```
s = 'IBM';
```

```
date = [datetime('now')-minutes(5) datetime('now')];
interval = '30S';

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

Display the first three rows of intraday data `d` for all valid Money.Net fields.

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

```
ans =

         Date              High      Low       Open      Close      Volume
    _________________     ______    ______    ______    ______    _______

    05/09/16 13:30:30     147.52    147.48    147.48    147.51    2763.00
    05/09/16 13:31:00     147.53    147.50    147.50    147.52    7241.00
    05/09/16 13:31:30     147.54    147.51    147.51    147.53    5608.00
```

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price
- Open price
- Close price
- Trading volume

Close the Money.Net connection.

```
close(c)
```

Retrieve Intraday Data in Minutes

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

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

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

Retrieve intraday data for yesterday in 30-minute bars for the symbol IBM using the Money.Net connection `c`. Specify the date as yesterday using `datetime`. To retrieve data in 30-minute bars, specify the interval as `'30M'`.

```
s = 'IBM';
date = datetime('yesterday');
interval = '30M';

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

Display the first three rows of intraday data `d` for all valid Money.Net fields.

```
d(1:3,:)

ans =

           Date             High      Low       Open      Close     Volume
    _________________    ______    ______    ______    ______    _______

    05/06/16 08:00:00    145.22    145.07    145.07    145.22    2455.00
    05/06/16 08:30:00    144.66    144.66    144.66    144.66     300.00
    05/06/16 09:00:00    145.00    144.90    144.90    145.00    4758.00
```

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price
- Open price
- Close price
- Trading volume

Close the Money.Net connection.

```
close(c)
```

Retrieve Daily Historical Data for Specified Fields

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

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

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

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

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

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

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

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

```
ans = 

           Date              High       Low  
    _________________       ______     ______

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

`d` is a table that contains these columns:

- Date timestamp
- High price
- Low price

Close the Money.Net connection.

```
close(c)
```

Retrieve Weekly Historical Data for Specified Fields

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

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

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

Retrieve historical data in weekly bars for the symbol IBM using the Money.Net connection `c`. Specify the date range from June 1, 2015 through June 30, 2015 using datetime. To retrieve weekly data, specify the interval as '7D'. Retrieve only the high and low price fields f from Money.Net.

```
s = 'IBM';
date = [datetime('1-Jun-2015') datetime('30-Jun-2015')];
interval = '7D';
f = {'High','Low'};

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

Display the first three rows of weekly data d.

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

```
ans = 

          Date             High      Low  
    _________________    ______    ______

    06/01/15 00:00:00    171.56    167.20
    06/08/15 00:00:00    170.44    163.37
    06/15/15 00:00:00    168.72    164.25
```

d is a table that contains these columns:

- Date timestamp
- High price
- Low price

Close the Money.Net connection.

```
close(c)
```

- "Retrieve Current and Historical Money.Net Data" on page 5-2

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using moneynet.

`s` — Money.Net symbol

character vector | cell array of character vector | string scalar

Money.Net symbol, specified as a character vector, cell array of a character vector, or a string scalar to denote one symbol.

Example: `'IBM'`

Data Types: `char`

`date` — Date

`datetime` array | character vector | cell array of character vectors | double | string array

Date, specified as a `datetime` array, character vector, cell array of character vectors, double, or a string array. If `date` contains one date, this date is the start date. The software determines the end date to be the last second of the same day. If `date` contains two dates, the first date is the start date and the second date is the end date.

Example: `datetime('yesterday')`

Data Types: `datetime` | `char` | `cell` | `double`

`interval` — Interval

character vector | string scalar

Interval between bars, specified as a character vector or string scalar. Specify the interval as a number followed by one of these letters: `S`, `M`, and `D`. These letters indicate seconds, minutes, and days, respectively. For example, `30S` is 30-second bars and `1D` is daily end-of-day data.

Data Types: `char`

`f` — Money.Net data field list

character vector | cell array of character vectors | string scalar | string array

Money.Net data field list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one field, use a character vector or string scalar. To specify multiple fields, use a cell array of character vectors or a string array.

Specify the field by using the single character or the field definition. For example, to specify the highest price for the equity during the current trading day, use a single character `'H'` or the corresponding field definition `'High'`. When using the field definition, the software ignores the case of the definition. To view the list of valid Money.Net fields and field definitions, see the Money.Net API Documentation.

Example: `'High'`

Example: `{'High','Low'}`

Data Types: `char`

Output Arguments

d — Money.Net data
table

Money.Net data, returned as a table. Each row in the table represents data at different times. The first column `Date` is the timestamp. The remaining columns contain one column of data for each Money.Net field `f`.

To return data for all available historical fields, use this syntax:

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

Money.Net returns data only for business days with trading activity.

See Also

See Also
`close` | `getdata` | `moneynet` | `news` | `realtime`

Topics
"Retrieve Current and Historical Money.Net Data" on page 5-2
"Money.Net Error and Warning Messages" on page 5-13

External Websites
Money.Net API Documentation

Introduced in R2016b

realtime

Retrieve Money.Net real-time data

Syntax

```
realtime(c,symbols)
realtime(c,symbols,eventhandler)
```

Description

`realtime(c,symbols)` subscribes to real-time data updates using the Money.Net connection `c` for the specified symbols. The default event handler function `mnRealTimeEventHandler` processes and retrieves real-time data updates for each specified symbol.

`realtime(c,symbols,eventhandler)` processes real-time data updates using a custom event handler function `eventhandler`.

Examples

Retrieve Money.Net Real-Time Data for One Symbol

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

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

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

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

```
symbol = 'IBM';

realtime(c,symbol)
```

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

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

Stop the symbol subscription.

```
stop(c)
```

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

Close the Money.Net connection.

```
close(c)
```

Retrieve Money.Net Real-Time Data for Multiple Symbols

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

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

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

Retrieve Money.Net real-time data updates for the symbols IBM and Yahoo!.

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

realtime(c,symbols)
```

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

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

Stop all symbol subscriptions.

```
stop(c)
```

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

Close the Money.Net connection.

```
close(c)
```

Retrieve Money.Net Real-Time Data Using a Custom Event Handler

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

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

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

Define a custom event handler function `myfcn`. The `myfcn` function displays real-time Money.Net data to the Command Window. You can write a custom function that processes real-time data updates differently. For details, see “Writing and Running Custom Event Handler Functions” on page 1-35.

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

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

```
symbol = 'IBM';

realtime(c,symbol,myfcn)
```

```
    Symbol              Description               Yesterday     YesterdayDateTime      Bid
    ______    _________________________________    _________    ___________________    _____

    'IBM'     'INTERNATIONAL BUSINESS MACHS'       148.31       05/24/16 00:00:00     151.6
```

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

Stop the symbol subscription.

```
stop(c)
```

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

Close the Money.Net connection.

```
close(c)
```

- “Retrieve Real-Time Money.Net Data” on page 5-6

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

symbols — Money.Net symbol list

character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char`

eventhandler — Event handler

`'mnRealTimeEventHandler'` (default) | character vector | string scalar | function handle

Event handler, specified as a character vector, string scalar, or a function handle that specifies the name of the event handler function. Write a custom event handler function to process any type of real-time Money.Net events. This function must have at least one input argument that is a table. The table format must be similar to the format of the output argument in `getdata`. The event handler function returns all available fields when it executes for the first time. The event handler function executes every time Money.Net provides a real-time update. For details about custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-35.

For example, to display real-time data updates in the Command Window, enter this code to define a custom event handler function:

```
symbol = 'IBM';
myfcn = @(x)disp(x);

realtime(c,symbol,myfcn)
```

If you do not specify a custom event handler function, the default event handler `mnRealTimeEventHandler` runs. To access the code for the default event handler, enter `edit mnRealTimeEventHandler.m`.

The `mnRealTimeEventHandler` function creates a workspace variable. The workspace variable name is a concatenation of the symbol name and the word `RealTime`. For example, `mnRealTimeEventHandler` populates real-time data for the symbol IBM into `IBMRealTime`. This workspace variable is a table with columns for each field. The values in the table change when Money.Net provides a real-time data update. Empty fields from Money.Net populate as `NaN`, `NaT`, and so on, depending on the data type.

First, `mnRealTimeEventHandler` runs using a table of current data. Then, `mnRealTimeEventHandler` runs each time an update occurs.

Data Types: `char` | `function_handle`

See Also

See Also
`close` | `getdata` | `getsubscriptions` | `moneynet` | `news` | `stop`

Topics
"Retrieve Real-Time Money.Net Data" on page 5-6
"Writing and Running Custom Event Handler Functions" on page 1-35
"Money.Net Error and Warning Messages" on page 5-13

External Websites
Money.Net API Documentation

Introduced in R2016b

stop

Unsubscribe Money.Net real-time data updates

Syntax

```
stop(c)
stop(c,symbols)
```

Description

`stop(c)` unsubscribes real-time data updates associated with the Money.Net connection `c`.

`stop(c,symbols)` unsubscribes real-time data updates for the specified symbols.

Examples

Unsubscribe All Real-Time Data Updates

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

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

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

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

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

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Unsubscribe from all symbol subscriptions.

```
stop(c)
```

The default event handler function stops processing all real-time data updates. For details about the event handler function, see `realtime`.

Close the Money.Net connection.

```
close(c)
```

Unsubscribe Real-Time Data Updates for One Symbol

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

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

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

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

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

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Unsubscribe from real-time data updates for IBM only.

```
symbol = 'IBM';

stop(c,symbol)
```

The default event handler function stops processing real-time data updates for IBM. The real-time data updates continue for Yahoo! only. For details about the event handler function, see `realtime`.

Close the Money.Net connection.

```
close(c)
```

- “Retrieve Real-Time Money.Net Data” on page 5-6

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

`symbols` — Money.Net symbol list
character vector | cell array of character vectors | string scalar | string array

Money.Net symbol list, specified as a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array.

Example: `'IBM'`

Example: `{'IBM','GOOG'}`

Data Types: `char`

See Also

See Also
`close` | `getsubscriptions` | `moneynet` | `realtime`

Topics
"Retrieve Real-Time Money.Net Data" on page 5-6
"Money.Net Error and Warning Messages" on page 5-13

External Websites
Money.Net API Documentation

Introduced in R2016b

getsubscriptions

Retrieve Money.Net subscribed symbols and event handler functions

Syntax

```
subs = getsubscriptions(c)
```

Description

`subs = getsubscriptions(c)` returns the subscription list `subs` that contains open subscriptions for the Money.Net connection `c`.

Examples

Retrieve Subscribed Symbols and Event Handlers

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

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

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

Subscribe to the symbols IBM and Yahoo! for real-time data updates using the Money.Net connection `c`.

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

realtime(c,symbols)
```

The default event handler function processes real-time data updates.

Retrieve the subscribed symbols and the corresponding event handler function for each symbol using the Money.Net connection `c`.

```
subs = getsubscriptions(c)
```

```
subs = 

    Symbols          EventHandlers      
    _______    ________________________

    'IBM'      @mnRealTimeEventHandler
    'YHOO'     @mnRealTimeEventHandler
```

`subs` returns a table with a row for each symbol and the corresponding event handler function.

Unsubscribe from all symbols using the Money.Net connection `c`.

```
stop(c)
```

Close the Money.Net connection.

```
close(c)
```

- "Retrieve Real-Time Money.Net Data" on page 5-6

Input Arguments

`c` — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

Output Arguments

`subs` — Subscription list

table

Subscription list, returned as a table. The list contains all currently subscribed symbols and the corresponding event handler function that is processing real-time updates for each symbol. Each row in the table represents one unique subscription.

If there are no subscribed symbols, `subs` is an empty table.

See Also

See Also
`close` | `moneynet` | `realtime` | `stop`

Topics
"Retrieve Real-Time Money.Net Data" on page 5-6
"Money.Net Error and Warning Messages" on page 5-13

External Websites
Money.Net API Documentation

Introduced in R2016b

optionchain

Retrieve Money.Net option symbols

Syntax

`o = optionchain(c,s)`

Description

`o = optionchain(c,s)` returns the option symbols using the Money.Net connection `c` and symbol `s`.

Examples

Retrieve Option Symbols for Specified Symbol

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

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

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

Retrieve option symbols `o` for the symbol IBM.

```
s = 'IBM';

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

`o` is a cell array of character vectors. Each character vector is an option symbol.

Display the first three option symbols.

```
o(1:3)

ans =
```

```
3×1 cell array

  'O:IBM\16Q13\130 .0'
  'O:IBM\16E27\148 .0'
  'O:IBM\16Q20\138 .0'
```

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

- Option symbol description
- Option symbol strike
- Option symbol expiration date

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

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

```
d =

          Symbol                   Description              Strike    Expiration
    ____________________    ___________________________    ______    __________

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

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

To retrieve intraday data, use `timeseries`.

Close the Money.Net connection.

```
close(c)
```

- "Retrieve Current and Historical Money.Net Data" on page 5-2

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

s — Money.Net symbol

character vector | cell array of character vector | string scalar

Money.Net symbol, specified as a character vector, cell array of a character vector, or a string scalar to denote one symbol.

Example: 'IBM'

Data Types: char

Output Arguments

o — Option symbols

cell array of character vectors

Option symbols, returned as a cell array of character vectors. Each character vector specifies one option symbol. The total number of option symbols depends on the symbol s.

See Also

See Also

close | getdata | moneynet | timeseries

Topics

"Retrieve Current and Historical Money.Net Data" on page 5-2
"Money.Net Error and Warning Messages" on page 5-13

External Websites

Money.Net API Documentation

Introduced in R2016b

news

Search and stream Money.Net latest news stories

Syntax

```
n = news(c)
n = news(c,Name,Value)
news(c,Name,Value)
```

Description

`n = news(c)` returns Money.Net news stories `n` using the Money.Net connection `c`.

`n = news(c,Name,Value)` returns news stories with additional options specified by one or more `Name,Value` pair arguments.

`news(c,Name,Value)` streams news stories in real time using the streaming options.

Examples

Retrieve News Stories

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

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

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

Retrieve news data `n` for 50 news stories using the Money.Net connection `c`.

```
n = news(c);
```

`n` returns as a table with 50 rows.

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

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

```
ans =

                              ArticleTitle                                   ArticleID 
    _________________________________________________________________________    __________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09
```

Close the Money.Net connection.

```
close(c)
```

Retrieve a Specific Number of Stories

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

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

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

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

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

n returns as a table with 10 rows.

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

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

```
ans =

                              ArticleTitle                                   ArticleID 
    _________________________________________________________________________    __________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09
```

Close the Money.Net connection.

```
close(c)
```

Filter News Story Retrieval by Specific Criteria

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

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

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

Retrieve news stories in the general finance category. Specify that the news stories mention the term `'Dropbox'` and contain the symbol for IBM.

```
category = 'General Finance';
term = 'Dropbox';
symbol = 'IBM';

n = news(c,'Category',category,'SearchTerm',term,'Symbol',symbol);
```

`n` is a table with one news story.

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

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

```
ans = 

                        ArticleTitle                         ArticleID      Publish
    ________________________________________________________    ___________    _______

    'Hewlett Packard Enterprise (HPE) Teams Up with Dropbox'    4.0002e+09     06/08/16
```

Close the Money.Net connection.

```
close(c)
```

Stream News Data

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

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

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

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

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

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

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

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

```
ans = 

                             ArticleTitle                                      ArticleID 
    _____________________________________________________________________    __________

    'Stop talking about replacements. Give PC owners something new al…'    3.8917e+09
```

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

Turn off the real-time news data stream.

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

Real-time updates stop in the workspace variable `mnNewsStreamLatest`.

Close the Money.Net connection.

```
close(c)
```

Stream News Data Using Custom Event Handler

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

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

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

Turn on the subscription to the Money.Net real-time news data stream using the custom event handler function `myfnc`. Here, define `myfnc` to display Money.Net news data to the Command Window. You can write a custom event handler function to process streaming news stories differently. For details, see "Writing and Running Custom Event Handler Functions" on page 1-35.

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

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

```
                              ArticleTitle                                     ArticleID
    _________________________________________________________________________    __________

    '@ETFcom: The Most Important ETF Of 2016 https://t.co/a5qCYK2o7c …'    3.9089e+09
```

Money.Net news stories stream to the Command Window.

Turn off the real-time news data stream.

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

Real-time updates stop in the Command Window.

Close the Money.Net connection.

```
close(c)
```

- "Retrieve Money.Net News Stories" on page 5-9

Input Arguments

c — Money.Net connection

connection object

Money.Net connection, specified as a connection object created using `moneynet`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

Note: The name-value pair arguments in the searching and streaming groups are independent. If you combine these name-value pair arguments, you receive this error:

```
Invalid combination of Name-Value pairs. Type HELP MONEYNET/NEWS to
see the valid syntax.
```

Searching News Stories Options

'Number' — Number of news stories

50 (default) | numeric scalar

Number of news stories, specified as the comma-separated pair consisting of 'Number' and a numeric scalar. The maximum number of news stories that the Money.Net API can return is 500.

The number of news stories returned can be fewer than the specified number because Money.Net provides only available news stories. When you specify this option by itself, news does not filter the story content.

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

Data Types: double

'SearchTerm' — Search term

character vector | string scalar

Search term, specified as the comma-separated pair consisting of 'SearchTerm' and a character vector or a string scalar. news returns available news stories that contain the search term in the title or body of the news story.

Example: `n = news(c,'SearchTerm','Windows 10');`

Data Types: char

'Symbol' — Symbol

character vector | cell array of character vectors | string scalar | string array

Symbol, specified as the comma-separated pair consisting of 'Symbol' and a character vector, cell array of character vectors, string scalar, or a string array. To specify one symbol, use a character vector or string scalar. To specify multiple symbols, use a cell array of character vectors or a string array. news returns news stories related to the specified symbols.

Example: `n = news(c,'Symbol',{'IBM,'YHOO'});`

Data Types: char | cell

`'Category'` — News category

character vector | string scalar

News category, specified as the comma-separated pair consisting of `'Category'` and a character vector or a string scalar. `news` returns stories only in the news category specified.

Example: `n = news(c,'Category','General Finance');`

Data Types: `char`

Streaming News Stories Options

`'Subscription'` — Money.Net real-time news subscription

`'on'` | `'off'`

Money.Net real-time news subscription, specified as the comma-separated pair consisting of `'Subscription'` and the values `'on'` or `'off'`. To turn on the Money.Net real-time news subscription, specify the value `'on'`. To turn off the subscription, specify the value `'off'`.

By default, the sample event handler function `mnNewsStreamEventHandler` processes the retrieval of news stories during real-time news subscription. To access the code for this function, enter `edit mnNewsStreamEventHandler.m`. The `mnNewsStreamEventHandler` function creates the workspace variable `mnNewsStreamLatest`. Then, `mnNewsStreamEventHandler` populates the table `mnNewsStreamLatest` with the latest 10 news stories from Money.Net. Then, the `mnNewsStreamEventHandler` function updates the list to display the latest news stories.

To specify a custom event handler function, use the name-value pair argument `'EventHandler'`.

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

Example: `news(c,'Subscription','on','EventHandler',myFcn)`

`'EventHandler'` — Custom event handler function

character vector | string scalar | function handle

Custom event handler function, specified as the comma-separated pair consisting of `'EventHandler'` and a character vector, string scalar, or function handle. To process the latest news stories, you can write your own custom event handler function. This function must have an input argument specified as a table. Each new news story from Money.Net is a single row in a table. For details about working with custom event handler functions, see “Writing and Running Custom Event Handler Functions” on page 1-35.

Specify this name-value pair argument only with the name-value pair argument `'Subscription'` and value `'on'`.

Example: `news(c,'Subscription','on','EventHandler',myFcn)`

Data Types: `char` | `function_handle`

Output Arguments

n — News stories
table

News stories, returned as a table with these variables. Each row in the table represents one news story. For details about these variables, see Money.Net API Documentation.

News Story Variable	Data Type	Variable Description
`ArticleTitle`	cell array of character vectors	News story title
`ArticleID`	double	Internal Money.Net identifier of the news story
`PublishedTime`	`datetime` array	Date and time the news story was published
`PublisherCode`	cell array of character vectors	Publisher four-digit code
`PublisherName`	cell array of character vectors	Publisher name
`ArticleBodyDescription`	cell array of character vectors	Body excerpt or short description of the news story
`Category`	cell array of character vectors	Internal Money.Net category of the news story
`URLLink`	cell array of character vectors	Website that contains the full news story

News Story Variable	Data Type	Variable Description
`Source`	cell array of character vectors	News stream source code
`Priority`	cell array of character vectors	Priority of news story with these values: • `0` is Normal • `1` is Breaking • `2` is Major Hot
`Symbols`	cell array	Symbols, or tickers, associated with the news story with these values: • An empty cell array for no symbols • A cell array that contains the symbol as a character vector for one symbol • A nested cell array that contains symbols as character vectors for multiple symbols

See Also

See Also

`close` | `getdata` | `moneynet` | `realtime` | `timeseries`

Topics

“Retrieve Money.Net News Stories” on page 5-9
“Writing and Running Custom Event Handler Functions” on page 1-35
“Money.Net Error and Warning Messages” on page 5-13

External Websites

Money.Net API Documentation

Introduced in R2016b

ravenpack

RavenPack News Analytics connection

Syntax

`c = ravenpack(username,password)`

Description

`c = ravenpack(username,password)` connects to RavenPack News Analytics Data Gateway using the user name `username` and password `password`.

Examples

Connect to RavenPack

Start the RavenPack Data Gateway process from the RavenPack folder in the Windows Start menu.

If you are running Linux®, start the RavenPack Data Gateway using this command. This command assumes that you installed RavenPack News Analytics in the default `/opt/ravenpack` folder.

```
/opt/ravenpack/bin/DataGateway.sh
```

Add the full path of the Data Gateway Client JAR file to the dynamic Java class path. For details about static and dynamic class paths, see “Java Class Path” (MATLAB).

```
javaaddpath 'C:\Program Files (x86)\RavenPack\api\DataGatewayClient.jar'
```

If you are running Linux, add the full path of the Data Gateway Client JAR file using this command.

```
javaaddpath '/opt/ravenpack/api/DataGatewayClient.jar'
```

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

```
c = ravenpack('username','pwd')
```

```
c = 

  ravenpack with properties:

    Application: 'rpna-api'
         Handle: [1x1 com.ravenpack.data.DataGatewayClient]
           User: 'username'
```

`c` is the RavenPack News Analytics connection object with these properties:

- RavenPack application
- RavenPack Data Gateway Client object
- User name

Close the RavenPack News Analytics connection.

```
close(c)
```

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

`username` — User name

character vector

User name, specified as a character vector to denote the name required to access RavenPack News Analytics.

Data Types: `char`

`password` — Password

character vector

Password, specified as a character vector to denote the password required to access RavenPack News Analytics.

Data Types: `char`

Output Arguments

c — RavenPack News Analytics connection
connection object

RavenPack News Analytics connection, returned as a connection object with these properties.

Property	Description
Application	RavenPack application service
Handle	RavenPack Data Gateway Client object
User	User name

See Also

See Also
`close` | `entitlements`

Topics
"Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22
"Workflow for RavenPack News Analytics" on page 3-26

External Websites
RavenPack Developer Zone Overview

Introduced in R2015b

entitlements

RavenPack News Analytics Data Gateway entitlements

Syntax

```
e = entitlements(c)
```

Description

`e = entitlements(c)` retrieves the RavenPack News Analytics entitlements using the RavenPack News Analytics connection `c`.

Examples

Retrieve RavenPack News Analytics Data Gateway Entitlements

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics Data Gateway entitlements `e`.

```
e = entitlements(c);
```

Display the first two rows and columns of the entitlement data.

```
e(1:2,1:2)
```

```
ans = 

          DISPLAYNAME                           NAME                 
    _________________________    __________________________________

    'RPNA-4.0-EQUITIES'          '(entity-scores :rpna-4.0-eqt)'
    'RPNA-4.0-GLOBAL-MACRO'      '(entity-scores :rpna-4.0-mgp)'
```

`e` is a table that contains the entitlement data. These columns specify the name of the RavenPack symbols. You can retrieve RavenPack News Analytics data for the RavenPack symbols specified in each row of the table.

For details about the column names in the table, see the RavenPack Developer Zone Overview.

List all the columns in the entitlement data.

```
e.Properties.VariableNames
```

```
ans =

  Columns 1 through 6

    'DISPLAYNAME'    'NAME'    'VERSION'    'SUFFIX'    'FIELDS'    'PRODSEQ'

  Column 7

    'PRODUCT'
```

Close the RavenPack News Analytics connection.

```
close(c)
```

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

Output Arguments

e — RavenPack News Analytics Data Gateway entitlements

table

RavenPack News Analytics Data Gateway entitlements, returned as a table.

Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

See Also

See Also
`close` | `ravenpack` | `timeseries`

Topics
"Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22
"Workflow for RavenPack News Analytics" on page 3-26

External Websites
RavenPack Developer Zone Overview

Introduced in R2015b

realtime

RavenPack News Analytics real-time data

Syntax

```
[status,lhandle] = realtime(c,symbol,listener)
```

Description

`[status,lhandle] = realtime(c,symbol,listener)` retrieves RavenPack News Analytics real-time data using the RavenPack News Analytics connection `c`.

Examples

Retrieve RavenPack News Analytics Real-Time Data

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics real-time data using RavenPack News Analytics connection `c`. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. Use the existing event listener `rpExampleListener`. Specify these fields as inputs to `rpExampleListener`:

- `ENTITY_NAME`
- `CATEGORY`
- `SUB_TYPE`

You can modify `rpExampleListener` or create your own event listener to add other functionality.

The MATLAB variable `evt` is an instance of a news event. Do not instantiate this variable. Assign this variable any name.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
```

```
fields = {'ENTITY_NAME','CATEGORY','SUB_TYPE'};

[status,lhandle] = realtime(c,symbol,...
                            @(~,evt)rpExampleListener(evt,fields))

status =

     1

lhandle =

 handle.listener
```

`status` returns a `1` to signify a successful RavenPack News Analytics connection.

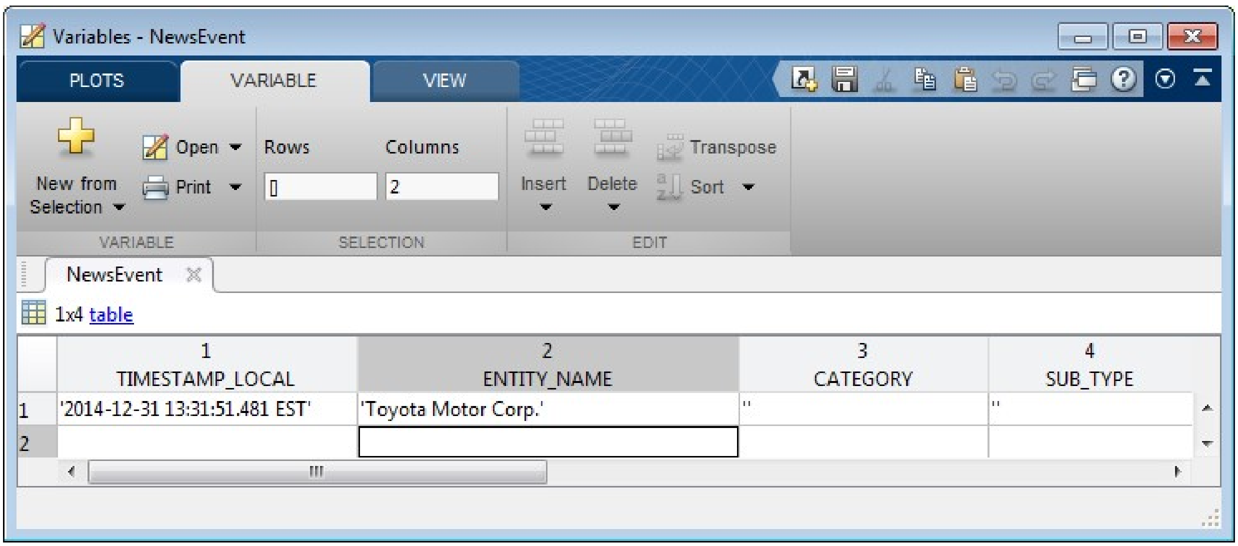
`lhandle` returns the contents of the handle to the RavenPack listener object.

The existing event listener `rpExampleListener` populates the MATLAB variable `NewsEvent` with the real-time data.

Display the real-time data.

```
openvar('NewsEvent')
```

MATLAB displays the Variables editor to show the news event data.



To stop real-time data updates, use the MATLAB `delete` function to delete the handle to the RavenPack listener object.

```
delete(lhandle)
```

Close the RavenPack News Analytics connection.

```
close(c)
```

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

`c` — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

`symbol` — RavenPack entitled symbol list

character vector | cell array of character vectors

RavenPack entitled symbol list, specified as a character vector or cell array of character vectors.

Data Types: `char` | `cell`

`listener` — Listener event handler

function handle

Listener event handler, specified as a function handle to listen for RavenPack News Analytics data events. You can modify the existing listener function `rpExampleListener` or create your own to add other functionality. You can find the code for the existing listener function in the `rpExampleListener.m` file.

Data Types: `function_handle`

Output Arguments

`status` — Subscription status

logical

Subscription status, returned as a logical `true` or `false`. `true` or `1` signifies a successful subscription to real-time RavenPack News Analytics data. `false` or `0` signifies an unsuccessful subscription to the real-time data or a subscription that is currently running.

`lhandle` — RavenPack News Analytics Gateway event listener
handle

RavenPack News Analytics Gateway event listener, returned as a handle to the RavenPack listener object.

Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

See Also

See Also
`close` | `entitlements` | `ravenpack` | `timeseries`

Topics
"Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22
"Workflow for RavenPack News Analytics" on page 3-26

External Websites
RavenPack Developer Zone Overview

Introduced in R2015b

rploader

RavenPack News Analytics file reader

Syntax

```
d = rploader(filename)
d = rploader(filename,Name,Value)
[d,h] = rploader( ___ )
```

Description

`d = rploader(filename)` reads the contents of the RavenPack News Analytics data file specified by `filename` and returns the contents in the MATLAB variable `d`.

`d = rploader(filename,Name,Value)` reads the contents of the file using additional options specified by one or more `Name,Value` pair arguments.

`[d,h] = rploader( ___ )` reads the contents of the file and retrieves the header information `h` in the file using any of the input arguments in the previous syntaxes.

Examples

Read Data in a File

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data.

```
filename = '2014-11-macro.csv';
```

```
d = rploader(filename);
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans = 

        TIMESTAMP_UTC          RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    ____________________    ____________    ___________    ___________

    24-Nov-2014 00:00:05    'F33A73'        'CMDT'         'Iron Ore'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about Iron Ore.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans = 

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'
```

```
  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Read Data Using a Name-Value Pair Argument

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. The RavenPack News Analytics entity name is Iron Ore.

```
filename = '2014-11-macro.csv';

d = rploader(filename,'entity_name',{'Iron Ore'});
```

`d` is a table that contains the RavenPack News Analytics data for Iron Ore.

Display the first four columns of the first record of data.

```
d(1,1:4)

ans =

        TIMESTAMP_UTC         RP_ENTITY_ID    ENTITY_TYPE    ENTITY_NAME
    ____________________    ____________    ___________    ___________

    24-Nov-2014 00:00:05    'F33A73'        'CMDT'         'Iron Ore'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about Iron Ore.

List the columns in the data.

```
d.Properties.VariableNames

ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Read Data Using Multiple Name-Value Pair Arguments

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. Read 5000 records in the data file starting at record number 10,000.

```
filename = '2014-11-macro.csv';
```

```
d = rploader(filename,'start',10000,'records',5000);
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans =

       TIMESTAMP_UTC         RP_ENTITY_ID    ENTITY_TYPE         ENTITY_NAME
    ____________________    ____________    ___________    ____________________

    24-Nov-2014 07:58:38    '0037F3'        'ORGA'         'State of Rajasthan'
```

`d` is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about the State of Rajasthan.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans =

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19
```

```
    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 34

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Read Data Within a Date Range

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Read the data from November 24, 2014 through November 25, 2014. Here, the file contains Equities News Analytics data.

```
filename = '2014-11-equities.csv';

d = rploader(filename,'date',{'11/24/2014','11/25/2014'});
```

`d` is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of data.

```
d(1,1:4)
```

```
ans =
```

```
    TIMESTAMP_UTC          RP_ENTITY_ID    ENTITY_TYPE           ENTITY_NAME
____________________    ____________    ___________    __________________________

24-Nov-2014 00:00:04    '355013'        'COMP'         'Panoramic Resources Ltd.'
```

d is a table with a header that contains the column names. There is one row in the table for each record of news data. Here, the first four columns specify a news event on November 24, 2014 about a company.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans = 

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'

  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 35
```

```
    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'    'COMPANY'

  Columns 36 through 43

    'ISIN'    'CSS'    'NIP'    'PEQ'    'BEE'    'BMQ'    'BAM'    'BCA'

  Columns 44 through 46

    'BER'    'ANL_CHG'    'MCQ'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Read Data and Return the Header Information

Open the RavenPack Data Feed Tool and create a RavenPack News Analytics data file. The resulting file is a comma-separated file. Open MATLAB. Navigate to the folder where the RavenPack News Analytics data file is located.

Read the data in the RavenPack News Analytics data file `filename`. Here, the file contains Global Macro News Analytics data. Read 5000 records in the data file starting at record number 10,000.

```
filename = '2014-11-macro.csv';

[d,h] = rploader(filename,'start',10000,'records',5000);
```

`d` is a table that contains the RavenPack News Analytics data.

`h` is a cell array that contains the header information.

Display the header information.

```
h
```

```
h =

    'TIMESTAMP_UTC'
    'RP_ENTITY_ID'
    'ENTITY_TYPE'
    'ENTITY_NAME'
    'POSITION_NAME'
    'RP_POSITION_ID'
    'COUNTRY_CODE'
```

```
'RELEVANCE'
'TOPIC'
'GROUP'
'TYPE'
'SUB_TYPE'
'PROPERTY'
'EVALUATION_METHOD'
'MATURITY'
'CATEGORY'
'ESS'
'AES'
'AEV'
'ENS'
'ENS_SIMILARITY_GAP'
'ENS_KEY'
'ENS_ELAPSED'
'G_ENS'
'G_ENS_SIMILARITY_GAP'
'G_ENS_KEY'
'G_ENS_ELAPSED'
'EVENT_SIMILARITY_KEY'
'NEWS_TYPE'
'SOURCE'
'RP_STORY_ID'
'RP_STORY_EVENT_INDEX'
'RP_STORY_EVENT_COUNT'
'PRODUCT_KEY'
```

For details, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

`filename` — RavenPack News Analytics data file

character vector

RavenPack News Analytics data file, specified as a character vector. To create this file, use the RavenPack Data Feed Tool to export news data into a comma-separated file.

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: `'date',{'11/24/2014'}`

`'date'` — Date or date range

cell array

Date or date range, specified as the comma-separated pair consisting of `'date'` and a cell array that contains one or two character vectors, date numbers, or `datetime` arrays. If you specify one date, `rploader` returns news data for the day specified by the character vector, date number, or `datetime` array in the cell array. To specify a date range, use `'date'` and a cell array that contains two character vectors, date numbers, or `datetime` arrays separated by a comma. The dates in the date range are inclusive.

Example: `'date',{'11/24/2014','11/25/2014'}`

Data Types: `cell`

`'rp_entity_id'` — RavenPack News Analytics entity identifier

cell array of character vectors

RavenPack News Analytics entity identifier, specified as the comma-separated pair consisting of `'rp_entity_id'` and a cell array of one or more character vectors. The character vectors denote the RavenPack entity identifiers. For details about the RavenPack entity identifiers, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Example: `'rp_entity_id',{'F33A73'}`

Data Types: `cell`

`'entity_name'` — RavenPack News Analytics entity name

cell array of character vectors

RavenPack News Analytics entity name, specified as a comma-separated pair consisting of `'entity_name'` and a cell array of one or more character vectors that denote RavenPack entity names. For details about the RavenPack entity names, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Example: `'entity_name',{'Iron Ore'}`

Data Types: `cell`

`'start'` — Reading offset
scalar

Reading offset, specified as a scalar to denote the record from which to start reading the RavenPack News Analytics data in the data file.

Example: `'start',100`

Data Types: `double`

`'records'` — Number of records to read
scalar

Number of records to read in the data file, specified as a scalar.

Example: `'records',5000`

Data Types: `double`

Output Arguments

d — RavenPack News Analytics data
table | double

RavenPack News Analytics data, returned as a table. If no matching data is found based on the specified name-value pair arguments, `d` returns as an empty double.

h — Header information
cell array

Header information, returned as a cell array. The header information contains the titles of each column in the returned data `d`.

See Also

See Also

`entitlements` | `ravenpack` | `realtime` | `timeseries`

Topics

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-22
“Workflow for RavenPack News Analytics” on page 3-26

External Websites

RavenPack Developer Zone Overview

Introduced in R2015b

timeseries

RavenPack News Analytics intraday and historical data

Syntax

```
d = timeseries(c,symbol,{startdate,enddate})
d = timeseries(c,symbol,{startdate,enddate},fields)
```

Description

`d = timeseries(c,symbol,{startdate,enddate})` retrieves RavenPack News Analytics intraday or historical data. This function uses the RavenPack News Analytics connection `c`, RavenPack entitled symbol, and a date range between the start date `startdate` and end date `enddate`.

`d = timeseries(c,symbol,{startdate,enddate},fields)` retrieves RavenPack News Analytics intraday or historical data for specific RavenPack fields.

Examples

Retrieve RavenPack News Analytics Intraday Data

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last 15 minutes. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is set to 15 minutes ago. The end date is the current date and time.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-.01;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

d is a table that contains the RavenPack News Analytics data.

Display the first four columns of the first record of intraday data.

```
d(1,1:4)
```

```
ans = 

       TIMESTAMP_UTC         RP_ENTITY_ID    ENTITY_TYPE     ENTITY_NAME 
    ____________________    ____________    ___________    _____________

    20-Jan-2015 15:52:30    '94A811'        'COMP'         'Novartis AG'
```

Each row in the table is one record of news data. Here, the first four columns specify a news event on January 20, 2015 about a company.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans = 

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 8

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'

  Columns 9 through 13

    'TOPIC'    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'

  Columns 14 through 19

    'EVALUATION_METHOD'    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'

  Columns 20 through 24

    'ENS'    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 27

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'
```

```
  Columns 28 through 31

    'EVENT_SIMILARITY...'    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'

  Columns 32 through 35

    'RP_STORY_EVENT_I...'    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'    'COMPANY'

  Columns 36 through 43

    'ISIN'    'CSS'    'NIP'    'PEQ'    'BEE'    'BMQ'    'BAM'    'BCA'

  Columns 44 through 46

    'BER'    'ANL_CHG'    'MCQ'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Close the RavenPack News Analytics connection.

```
close(c)
```

Retrieve RavenPack News Analytics Intraday Data with Fields

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last minute using RavenPack fields. Here, the symbol is set to the entitled symbol `(entity-scores :rpna-4.0-eqt)`. The start date is set to a minute ago. The end date is the current date and time. The fields list contains these fields:

- `ENTITY_NAME`
- `CATEGORY`
- `SUB_TYPE`

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-.001;
enddate = now;
```

```
fields = {'ENTITY_NAME','CATEGORY','SUB_TYPE'};

d = timeseries(c,symbol,{startdate,enddate},fields)

d = 

       TIMESTAMP_UTC                      ENTITY_NAME                  CATEGORY    SUB_TYPE
    ____________________    __________________________________    ________    ________

    31-Dec-2014 18:18:39    'Ernst & Young LLP'                     []          []
...
```

d is a table that contains RavenPack News Analytics data for the companies with news events. Each row in the table is a news event for a company.

Close the RavenPack News Analytics connection.

```
close(c)
```

Retrieve RavenPack News Analytics Historical Data

Create a RavenPack News Analytics connection c using the user name username and password pwd.

```
c = ravenpack('username','pwd');
```

Retrieve RavenPack News Analytics data for the last day. Here, the symbol is set to the entitled symbol (entity-scores :rpna-4.0-eqt). The start date is set to 1 day ago. The end date is the current date and time.

```
symbol = '(entity-scores :rpna-4.0-eqt)';
startdate = now-1;
enddate = now;

d = timeseries(c,symbol,{startdate,enddate});
```

d is a table that contains the RavenPack News Analytics data.

To retrieve more than 3 days of historical news data, use the RavenPack News Analytics Data Feed Tool.

Display the first four columns of the first record of historical data.

```
d(1,1:4)
```

```
ans = 

       TIMESTAMP_UTC          RP_ENTITY_ID    ENTITY_TYPE             ENTITY_NAME
    ____________________     ____________    ___________    ____________________________

    19-Jan-2015 16:21:00     'D5E3FF'        'COMP'         'Tekmira Pharmaceuticals Co
```

Each row in the table is one record of news data. Here, the first four columns specify a news event on January 19, 2015 about a company.

List the columns in the data.

```
d.Properties.VariableNames
```

```
ans = 

  Columns 1 through 4

    'TIMESTAMP_UTC'    'RP_ENTITY_ID'    'ENTITY_TYPE'    'ENTITY_NAME'

  Columns 5 through 9

    'POSITION_NAME'    'RP_POSITION_ID'    'COUNTRY_CODE'    'RELEVANCE'    'TOPIC'

  Columns 10 through 14

    'GROUP'    'TYPE'    'SUB_TYPE'    'PROPERTY'    'EVALUATION_METHOD'

  Columns 15 through 20

    'MATURITY'    'CATEGORY'    'ESS'    'AES'    'AEV'    'ENS'

  Columns 21 through 24

    'ENS_SIMILARITY_GAP'    'ENS_KEY'    'ENS_ELAPSED'    'G_ENS'

  Columns 25 through 28

    'G_ENS_SIMILARITY...'    'G_ENS_KEY'    'G_ENS_ELAPSED'    'EVENT_SIMILARITY...'

  Columns 29 through 32

    'NEWS_TYPE'    'SOURCE'    'RP_STORY_ID'    'RP_STORY_EVENT_I...'

  Columns 33 through 38
```

```
    'RP_STORY_EVENT_C...'    'PRODUCT_KEY'    'COMPANY'    'ISIN'    'CSS'    'NIP'

  Columns 39 through 46

    'PEQ'    'BEE'    'BMQ'    'BAM'    'BCA'    'BER'    'ANL_CHG'    'MCQ'
```

For details about each column in the table, see *RavenPack News Analytics User Guide and Service Overview* in the RavenPack Developer Zone Overview.

Close the RavenPack News Analytics connection.

```
close(c)
```

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

symbol — RavenPack entitled symbol

character vector | cell array of character vectors

RavenPack entitled symbol, specified as a character vector or cell array with one character vector.

Data Types: `char` | `cell`

startdate — Start date

numeric scalar | character vector | `datetime` array

Start date, specified as a numeric scalar, character vector, or `datetime` array to denote the start date of the date range for the returned news data.

Example: `now-.01`

Data Types: `double` | `char` | `datetime`

enddate — End date

numeric scalar | character vector | datetime array

End date, specified as a numeric scalar, character vector, or `datetime` array to denote the end date of the date range for the returned news data.

Example: `now`

Data Types: `double` | `char` | `datetime`

fields — RavenPack field list

character vector | cell array of character vectors

RavenPack fields list, specified as a character vector or cell array of one or more character vectors. Each character vector corresponds to a RavenPack field. The fields determine the news data to return. For details about the fields, contact RavenPack.

Example: `{'ENTITY_NAME','CATEGORY','SUB_TYPE'}`

Data Types: `char` | `cell`

Output Arguments

d — RavenPack News Analytics data

table

RavenPack News Analytics data, specified as a table.

Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.
- If you encounter this error, decrease the date range using the input arguments `startdate` and `enddate`.

  ```
  Java exception occurred:
  ```

```
com.ravenpack.data.DataGatewayException: TIMEOUT
while invoking remote function:
jl-api.getQuoteRange
```

See Also

See Also

`close` | `entitlements` | `ravenpack` | `realtime`

Topics

“Determine the Event Volume Indicator Using RavenPack News Analytics” on page 3-22
“Workflow for RavenPack News Analytics” on page 3-26

External Websites

RavenPack Developer Zone Overview

Introduced in R2015b

close

Close RavenPack News Analytics connection

Syntax

```
close(c)
```

Description

`close(c)` closes the RavenPack News Analytics connection `c`.

Examples

Close the RavenPack News Analytics Connection

Create a RavenPack News Analytics connection `c` using the user name `username` and password `pwd`.

```
c = ravenpack('username','pwd');
```

Close the RavenPack News Analytics connection.

```
close(c)
```

- "Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22

Input Arguments

c — RavenPack News Analytics connection

connection object

RavenPack News Analytics connection, specified as a connection object created using `ravenpack`.

Tips

- Before creating a RavenPack News Analytics connection:
  - Start the RavenPack Data Gateway process.
  - Add the full path of the Data Gateway Client JAR file to the static or dynamic Java class path.

  For details, see `ravenpack`.

See Also

See Also
`ravenpack`

Topics
"Determine the Event Volume Indicator Using RavenPack News Analytics" on page 3-22
"Workflow for RavenPack News Analytics" on page 3-26

External Websites
RavenPack Developer Zone Overview

Introduced in R2015b

elektron

Thomson Reuters Elektron Message API connection

Description

The `elektron` function creates an `elektron` object. The `elektron` object represents a Thomson Reuters Elektron connection.

Retrieve current and real-time data using an `elektron` object. You can retrieve data based on your credentials that consist of a user name and custom IP address. For details about credentials, contact Thomson Reuters Elektron.

When you install Thomson Reuters Elektron on your computer, the installation folder contains JAR files. Add these JAR files to the dynamic Java class path every time you connect to Thomson Reuters Elektron:

- `ansipage.jar`
- `ema-javadoc.jar`
- `ema.jar`
- `jdacsUpalib.jar`
- `upa.jar`
- `upaValueAdd.jar`
- `upaValueAddCache.jar`
- `commons-configuration-1.10.jar`
- `commons-lang-2.6.jar`
- `commons-logging-1.2.jar`
- `org.apache.commons.collections.jar`
- `slf4j-api-1.7.12.jar`
- `slf4j-jdk14-1.7.12.jar`

Alternatively, you can add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Create Object

Syntax

Description

`c = elektron(username,service)` creates a Thomson Reuters Elektron connection using the Thomson Reuters user name and service name.

`c = elektron(username,service,ipaddress)` specifies the IP address for the connection.

`c = elektron(username,service,ipaddress,portnumber)` specifies the port number for the connection.

Input Arguments

`username` — Thomson Reuters user name
character vector | string

Thomson Reuters user name, specified as a character vector or string. To find your user name, contact Thomson Reuters.

Data Types: `char` | `string`

`service` — Thomson Reuters service name
character vector | string

Thomson Reuters service name, specified as a character vector or string. This input argument indicates the enabled Thomson Reuters Elektron service for your Thomson Reuters user name. To find your service name, contact Thomson Reuters.

Data Types: `char` | `string`

`ipaddress` — IP address for Thomson Reuters server
`'localhost'` (default) | character vector | string

IP address for Thomson Reuters server where Thomson Reuters Elektron data is available, specified as a character vector or string. To find the IP address for the Thomson Reuters server, contact Thomson Reuters.

Data Types: `char` | `string`

`portnumber` — Port number
`14002` (default) | numeric scalar

Port number for Thomson Reuters Elektron connection, specified as a numeric scalar. To find the port number, contact Thomson Reuters.

Data Types: `double`

Properties

`IPAddress` — IP address for Thomson Reuters server
`'localhost'` (default) | character vector | string

IP address for Thomson Reuters server where Thomson Reuters Elektron data is available, specified as a character vector or string. To find the IP address for the Thomson Reuters server, contact Thomson Reuters.

Data Types: `char` | `string`

`Port` — Port number
`14002` (default) | numeric scalar

Port number for Thomson Reuters Elektron connection, specified as a numeric scalar. To find the port number, contact Thomson Reuters.

Data Types: `double`

`Service` — Thomson Reuters service name
character vector | string

Thomson Reuters service name, specified as a character vector or string. This input argument indicates the enabled Thomson Reuters Elektron service for your Thomson Reuters user name. To find your service name, contact Thomson Reuters.

Data Types: `char` | `string`

`Username` — Thomson Reuters user name
character vector | string

Thomson Reuters user name, specified as a character vector or string. To find your user name, contact Thomson Reuters.

Data Types: char | string

Object Functions

close	Close Thomson Reuters Elektron connection
getdata	Thomson Reuters Elektron current market data
realtime	Thomson Reuters Elektron real-time market data

Examples

Create Thomson Reuters Elektron Connection

First, create a Thomson Reuters Elektron connection. Then, retrieve current market data. Close the connection. The current market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)

 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Thomson Reuters Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Thomson Reuters Elektron field identifier
- `DataType` — Thomson Reuters Elektron data type of the Thomson Reuters Elektron field
- `Name` — Thomson Reuters Elektron field name
- `Value` — Current Thomson Reuters Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)

d =

  284×4 table array

    FieldId    DataType        Name                  Value
    _______    ________    _____________    ______________________

    [    1]    [18]        'PROD_PERM'      '62'
    [    2]    [18]        'RDNDISPLAY'     '67'
    [    3]    [31]        'DSPLY_NAME'     'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)

ans =
```

```
  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

Create Thomson Reuters Elektron Connection Using IP Address

First, create a Thomson Reuters Elektron connection using an IP address. Then, retrieve current market data. Close the connection. The current market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name, service name, and IP address.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
```

```
ipaddress = '123.123.123.123';

c = elektron(username,servicename,ipaddress)

 c = 

 elektron with properties:
    IPAddress: '123.123.123.123'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Thomson Reuters Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Thomson Reuters Elektron field identifier
- `DataType` — Thomson Reuters Elektron data type
- `Name` — Thomson Reuters Elektron field name
- `Value` — Current Thomson Reuters Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)

d = 

  284×4 table array

    FieldId     DataType        Name                 Value          
    _______     ________    _____________    ______________________

    [    1]     [18]        'PROD_PERM'      '62'                  
    [    2]     [18]        'RDNDISPLAY'     '67'                  
    [    3]     [31]        'DSPLY_NAME'     'DELAYED-15INTL B'    
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)

ans = 

  3×1 cell array
```

```
'PROD_PERM'
'RDNDISPLAY'
'DSPLY_NAME'
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

Create Thomson Reuters Elektron Connection Using Port Number

First, create a Thomson Reuters Elektron connection using a port number. Then, retrieve current market data. Close the connection. The current market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name, service name, IP address, and port number.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';
ipaddress = '123.123.123.123';
portnumber = '1234';
```

```
c = elektron(username,servicename,ipaddress,portnumber)
```

```
 c =

 elektron with properties:
    IPAddress: '123.123.123.123'
         Port: 1234
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Thomson Reuters Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Thomson Reuters Elektron field identifier
- `DataType` — Thomson Reuters Elektron data type
- `Name` — Thomson Reuters Elektron field name
- `Value` — Current Thomson Reuters Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d = 

  284×4 table array

    FieldId    DataType        Name                  Value
    _______    ________    ____________    ______________________

    [    1]    [18]        'PROD_PERM'     '62'
    [    2]    [18]        'RDNDISPLAY'    '67'
    [    3]    [31]        'DSPLY_NAME'    'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans = 

  3×1 cell array
```

```
    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

See Also

Topics

“Decide to Buy Shares Using Thomson Reuters Elektron Current Data” on page 6-2
“Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data” on page 6-5

External Websites

Thomson Reuters Elektron

Introduced in R2017a

getdata

Thomson Reuters Elektron current market data

Syntax

```
d = getdata(c,s)
```

Description

`d = getdata(c,s)` returns Thomson Reuters Elektron current market data using the Thomson Reuters Elektron connection for a single security.

Examples

Retrieve Current Thomson Reuters Elektron Data

First, create a Thomson Reuters Elektron connection. Then, retrieve current market data. Close the connection. The current market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)
```

```
 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Thomson Reuters Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Thomson Reuters Elektron field identifier
- `DataType` — Thomson Reuters Elektron data type of the Thomson Reuters Elektron field
- `Name` — Thomson Reuters Elektron field name
- `Value` — Current Thomson Reuters Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array

    FieldId    DataType        Name                  Value
    _______    ________    ____________    ______________________

    [    1]    [18]        'PROD_PERM'     '62'
    [    2]    [18]        'RDNDISPLAY'    '67'
    [    3]    [31]        'DSPLY_NAME'    'DELAYED-15INTL B'
```

```
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)
```

```
ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

- “Decide to Buy Shares Using Thomson Reuters Elektron Current Data” on page 6-2

Input Arguments

c — Thomson Reuters Elektron connection

`elektron` object

Thomson Reuters Elektron connection, specified as an `elektron` object created using the `elektron` function.

s — Thomson Reuters Elektron security

character vector | string

Thomson Reuters Elektron security, specified as a character vector or string. You can specify only a single security. The security is a Reuters Instrument Code (RIC). For details about RICs, contact Thomson Reuters.

Example: `'IBM.N'`

Data Types: `char` | `string`

Output Arguments

d — Thomson Reuters Elektron current market data

table

Thomson Reuters Elektron current market data, specified as a table. The `getdata` function returns Thomson Reuters Elektron current market data for the specified security `s`.

Variable	Data Type	Description
`FieldId`	Cell array of doubles	Thomson Reuters Elektron field identifier
`DataType`	Cell array of doubles	Thomson Reuters Elektron data type of the Thomson Reuters Elektron field
`Name`	Cell array of character vectors	Thomson Reuters Elektron field name
`Value`	Cell array of doubles or character vectors	Current Thomson Reuters Elektron data value

MATLAB converts the current data value from Thomson Reuters Elektron as follows:

- Numeric data values convert to doubles
- Text values convert to character vectors
- Date and time values convert to character vectors

For details about fields and their availability, see Thomson Reuters Elektron.

See Also

See Also

`close` | `elektron` | `realtime`

Topics

"Decide to Buy Shares Using Thomson Reuters Elektron Current Data" on page 6-2

External Websites

Thomson Reuters Elektron

Introduced in R2017a

realtime

Thomson Reuters Elektron real-time market data

Syntax

```
reqid = realtime(c,seclist,eventhandler)
reqid = realtime(c,r,eventhandler)
```

Description

`reqid = realtime(c,seclist,eventhandler)` returns a real-time request identifier using the Thomson Reuters Elektron connection, security list, and an event handler function. You can retrieve Thomson Reuters Elektron real-time market data by accessing variables that appear in the MATLAB workspace.

`reqid = realtime(c,r,eventhandler)` retrieves Thomson Reuters Elektron real-time market data using a Thomson Reuters Elektron custom request message.

Examples

Retrieve Thomson Reuters Elektron Real-Time Data

First, create a Thomson Reuters Elektron connection. Then, retrieve real-time market data. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
```

```
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see "Java Class Path" (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

Retrieve real-time market data for the IBM security using the Thomson Reuters Elektron connection. Use the example event handler function `elektronExampleListener`. To access the code for this function, enter `edit elektronExampleListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Thomson Reuters Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
seclist = 'IBM.N';
eventhandler = @(~,ev)elektronExampleListener(ev);
reqid = realtime(c,seclist,eventhandler)
```

```
reqid = 

  struct with fields:

      ReqId: 5
     ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
     Handle: [1×1 datafeedElektron]
```

```
    Listener: [1×1 handle.listener]
```

The cell array `IBM` appears in the MATLAB workspace. `IBM` contains four columns. The columns are:

- Thomson Reuters Elektron field identifier
- Thomson Reuters Elektron field name
- Thomson Reuters Elektron field data type
- Thomson Reuters Elektron field real-time data value

Access real-time data in the cell array. Access the Thomson Reuters Elektron field name in the second column and the current data value in the fourth column.

```
field = IBM{1,2};
value = IBM{1,4};
```

For events that do not have an event name or that are not associated with the security, the `ElektronExampleData` object appears in the MATLAB workspace. This object contains decoded event data. View the contents of this object. The contents of this object vary depending on the Thomson Reuters Elektron event.

```
ElektronExampleData
```

```
ElektronExampleData =

UpdateMsg
    streamId="10"
    domain="MarketPrice Domain"
    updateTypeNum="1"
    Payload dataType="FieldList"
        FieldList
            FieldEntry fid="22" name="BID" dataType="Real" value="57.1"
            FieldEntry fid="25" name="ASK" dataType="Real" value="57.11"
            FieldEntry fid="30" name="BIDSIZE" dataType="Real" value="22.0"
            FieldEntry fid="31" name="ASKSIZE" dataType="Real" value="39.0"
            FieldEntry fid="11683" name="BIDFINMMID" dataType="Rmtes" value="(blank dat
            FieldEntry fid="11684" name="ASKFINMMID" dataType="Rmtes" value="(blank dat
            FieldEntry fid="3298" name="BIDXID" dataType="Enum" value="43"
            FieldEntry fid="3297" name="ASKXID" dataType="Enum" value="6"
            FieldEntry fid="6579" name="BID_COND_N" dataType="Rmtes" value="R"
            FieldEntry fid="6580" name="ASK_COND_N" dataType="Rmtes" value="R"
            FieldEntry fid="293" name="BID_MMID1" dataType="Rmtes" value="NAS"
            FieldEntry fid="296" name="ASK_MMID1" dataType="Rmtes" value="XPH"
```

```
            FieldEntry fid="1000" name="GV1_TEXT" dataType="Rmtes" value="A"
            FieldEntry fid="8937" name="LIMIT_INDQ" dataType="Enum" value="25"
            FieldEntry fid="3887" name="SEQNUM_QT" dataType="Real" value="1.6923329E7"
            FieldEntry fid="118" name="PRC_QL_CD" dataType="Enum" value="0"
            FieldEntry fid="3264" name="PRC_QL3" dataType="Enum" value="0"
            FieldEntry fid="8406" name="QTE_ORIGIN" dataType="Rmtes" value=" "
            FieldEntry fid="1041" name="GV1_FLAG" dataType="Rmtes" value=" "
            FieldEntry fid="12783" name="NBBO_IND" dataType="Enum" value="5"
            FieldEntry fid="3855" name="QUOTIM_MS" dataType="UInt" value="62664591"
            FieldEntry fid="1025" name="QUOTIM" dataType="Time" value="17:24:24:000:000
            FieldEntry fid="14238" name="ORDRECV_MS" dataType="Time" value="17:24:24:59
            FieldEntry fid="14246" name="ORDREC2_MS" dataType="Time" value="(blank data
        FieldListEnd
    PayloadEnd
UpdateMsgEnd
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

Retrieve Thomson Reuters Elektron Real-Time Last Trade Price Data

First, create a Thomson Reuters Elektron connection. Then, retrieve real-time last trade price data. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
```

```
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

Retrieve real-time market data for the IBM security using the Thomson Reuters Elektron connection. Use the example event handler function `elektronPriceTableListener`. To access the code for this function, enter `edit elektronPriceTableListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Thomson Reuters Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
seclist = 'IBM.N';
eventhandler = @(~,ev)elektronPriceTableListener(ev,seclist);
reqid = realtime(c,seclist,eventhandler)

reqid = 

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The table `PriceTable` appears in the MATLAB workspace. `PriceTable` contains these variables:

- `RIC` — RIC for the IBM security
- `TRDPRC_1` — Last trade price data

Thomson Reuters Elektron continuously updates the `TRDPRC_1` variable with the latest trade price in the table row.

Access the last trade price in real time for the IBM security.

```
PriceTable.TRDPRC_1
```

```
ans =

  163.2600
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

Retrieve Thomson Reuters Elektron Real-Time Data Using Custom Request Message

First, create a Thomson Reuters Elektron connection. Then, retrieve real-time data using a custom request message. Close the connection. The real-time market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename);
```

Create a custom request message for the Apple security. Enter this Thomson Reuters Elektron Message API code.

```
import com.thomsonreuters.ema.access.*;
import com.thomsonreuters.ema.access.OmmConsumerConfig.*;

r = EmaFactory.createReqMsg;
r.serviceName(c.Service).name('AAPL.O');
```

Retrieve last trade price data using the Thomson Reuters Elektron connection and custom request message. Use the example event handler function `elektronExampleListener`. To access the code for this function, enter `edit elektronExampleListener.m`.

`reqid` is a structure that contains these fields:

- `ReqId` — Request identifier for the real-time data request
- `ReqMsg` — Thomson Reuters Elektron Message API request object
- `Handle` — MATLAB event listener process object
- `Listener` — MATLAB event listener object

```
eventhandler = @(~,ev)elektronExampleListener(ev);
reqid = realtime(c,r,eventhandler)
```

```
reqid = 

  struct with fields:

       ReqId: 5
      ReqMsg: [1×1 com.thomsonreuters.ema.access.ReqMsgImpl]
      Handle: [1×1 datafeedElektron]
    Listener: [1×1 handle.listener]
```

The cell array AAPL appears in the MATLAB workspace.

Access real-time data in the cell array. Access the Thomson Reuters Elektron field name in the second column and the current data value in the fourth column.

```
field = AAPL{1,2};
value = AAPL{1,4};
```

Stop real-time data subscription.

```
delete(reqid.Listener)
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

- "Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data" on page 6-5

Input Arguments

c — Thomson Reuters Elektron connection

elektron object

Thomson Reuters Elektron connection, specified as an `elektron` object created using the `elektron` function.

seclist — Thomson Reuters Elektron security list

character vector | string | cell array of character vectors | string array

Thomson Reuters Elektron security list, specified as a character vector, string, cell array of character vectors, or string array. Specify securities in the security list as:

- A character vector or string for one security.
- A cell array of character vectors or a string array for multiple securities. Each character vector in the cell array or each string in the string array is a Thomson Reuters Elektron security.

The security is a Reuters Instrument Code (RIC). For details about RICs, contact Thomson Reuters.

Example: `{'IBM.N','MSFT.O'}`

Data Types: `cell` | `char` | `string`

`r` — Thomson Reuters Elektron request message

Thomson Reuters Elektron Message API request object

Thomson Reuters Elektron request message, specified as a Thomson Reuters Elektron Message API request object. To create this object, use the Thomson Reuters Elektron Message API code. For details, see Thomson Reuters Elektron. This object defines the data and parameters for a custom message request.

`eventhandler` — Event handler

function handle

Event handler, specified as a function handle. You can use the example event handling functions `elektronExampleListener` or `elektronPriceTableListener` to process real-time Thomson Reuters Elektron Message API events. Or, you can define a custom event handler function to process events of your choice. For details, see “Writing and Running Custom Event Handler Functions” on page 1-35.

The event handler function `elektronExampleListener` creates a cell array for each security in the input argument `seclist`. The cell array contains real-time data for the security. For example, if `seclist` contains `'IBM.N'`, then the cell array named `IBM` appears in the MATLAB workspace.

For events without a name or that are not associated with a RIC, the event handler function creates an object named `ElektronExampleData` in the MATLAB workspace. This object contains decoded data for these events. The object type differs depending on the current Thomson Reuters Elektron Message API output. For details about this object, see Thomson Reuters Elektron.

This table describes the columns in the cell arrays for each security that appear in the MATLAB workspace.

Cell Array Column	Description
First column	Thomson Reuters Elektron field identifier
Second column	Thomson Reuters Elektron field name
Third column	Thomson Reuters Elektron data type of the Thomson Reuters Elektron field
Fourth column	Current Thomson Reuters Elektron data value

The event handler function `elektronExampleListener` has an input argument `ev` that is specified as a four-column cell array that contains:

- Decoded data
- Thomson Reuters Elektron Message API event object
- Event message
- Event name

The other event handler function `elektronPriceTableListener` creates a table named `PriceTable` in the MATLAB workspace. This table contains the last trade price for each security in the input argument `seclist`. The table contains two variables, `RIC` for the security and `TRDPRC_1` for the last trade price of the corresponding security:

- `RIC` is returned as a cell array of one or more character vectors that depend on the number of securities in the security list.
- `TRDPRC_1` is returned as a numeric scalar for one security or an array of doubles that has a value for each security in `RIC`.

The event handler function `elektronPriceTableListener` has the security list as an additional input argument. The security list matches the contents of `seclist`.

To access the code for the event handler function `elektronExampleListener`, enter `edit elektronExampleListener.m`. To access the code for the event handler function `elektronPriceTableListener`, enter `edit elektronPriceTableListener.m`.

For example, to retrieve real-time data for the IBM and Microsoft securities, enter this code that assumes an established Thomson Reuters Elektron connection `c`.

```
seclist = {'IBM.N','MSFT.O'};
reqId = realtime(c,seclist,@(~,ev)elektronExampleListener(ev));
```

Example: `@(~,ev)elektronExampleListener(ev)`

Data Types: `function_handle`

Output Arguments

`reqid` — Real-time request identifier

structure

Real-time request identifier, returned as a structure with these fields.

Field	Description
`ReqId`	Request identifier for the real-time data request
`ReqMsg`	Thomson Reuters Elektron Message API request object
`Handle`	MATLAB event listener process object
`Listener`	MATLAB event listener object

For details about the request identifier and request object, see Thomson Reuters Elektron.

For details about MATLAB event listeners, see "Overview Events and Listeners" (MATLAB).

Use the real-time request identifier to stop the real-time data subscription; for example:

```
delete(reqid.Listener)
```

Tips

- If you encounter issues with Thomson Reuters Elektron data retrieval, see the contents of the `ElektronExampleData` object in the MATLAB workspace. For example, if you enter an invalid RIC, the `ElektronExampleData` object contains a status message that specifies the RIC cannot be found. To retrieve such information automatically, write a custom event handler function that parses status messages in the `ElektronExampleData` object. For details about custom event handler functions, see "Writing and Running Custom Event Handler Functions" on page 1-35. For details about the status messages, see Thomson Reuters Elektron.

See Also

See Also
`close` | `elektron` | `getdata`

Topics
"Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data" on page 6-5

"Writing and Running Custom Event Handler Functions" on page 1-35

External Websites

Thomson Reuters Elektron

Introduced in R2017a

close

Close Thomson Reuters Elektron connection

Syntax

`close(c)`

Description

`close(c)` closes the Thomson Reuters Elektron connection.

Examples

Close Thomson Reuters Elektron Connection

First, create a Thomson Reuters Elektron connection. Then, retrieve current market data. Close the connection. The current market data you see when running this code can differ from the output data here.

Add Thomson Reuters Elektron JAR files to the dynamic Java class path. Find these JAR files in the installation folder. Here, the installation folder is `i:\Elektron`.

```
javaaddpath i:\Elektron\ansipage.jar
javaaddpath i:\Elektron\ema-javadoc.jar
javaaddpath i:\Elektron\ema.jar
javaaddpath i:\Elektron\jdacsUpalib.jar
javaaddpath i:\Elektron\upa.jar
javaaddpath i:\Elektron\upaValueAdd.jar
javaaddpath i:\Elektron\upaValueAddCache.jar
javaaddpath i:\Elektron\commons-configuration-1.10.jar
javaaddpath i:\Elektron\commons-lang-2.6.jar
javaaddpath i:\Elektron\commons-logging-1.2.jar
javaaddpath i:\Elektron\org.apache.commons.collections.jar
javaaddpath i:\Elektron\slf4j-api-1.7.12.jar
javaaddpath i:\Elektron\slf4j-jdk14-1.7.12.jar
```

Alternatively, add these JAR files to the static Java class path. For details about dynamic and static class paths, see “Java Class Path” (MATLAB).

Connect to Thomson Reuters Elektron using the user name and service name.

`c` is an `elektron` object.

```
username = 'username';
servicename = 'servicename';

c = elektron(username,servicename)
```

```
 c =

 elektron with properties:
    IPAddress: 'localhost'
         Port: 14002
      Service: 'servicename'
     Username: 'username'
```

Retrieve current data for the IBM security using the Thomson Reuters Elektron connection.

`d` is a table that contains the current data. The variables are:

- `FieldId` — Thomson Reuters Elektron field identifier
- `DataType` — Thomson Reuters Elektron data type of the Thomson Reuters Elektron field
- `Name` — Thomson Reuters Elektron field name
- `Value` — Current Thomson Reuters Elektron data value

```
s = 'IBM.N';
d = getdata(c,s)
```

```
d =

  284×4 table array

    FieldId    DataType         Name                   Value
    _______    ________    ____________    ______________________

    [    1]    [18]        'PROD_PERM'     '62'
    [    2]    [18]        'RDNDISPLAY'    '67'
    [    3]    [31]        'DSPLY_NAME'    'DELAYED-15INTL B'
    ...
```

Access the first three field names in the current data.

```
d.Name(1:3)

ans =

  3×1 cell array

    'PROD_PERM'
    'RDNDISPLAY'
    'DSPLY_NAME'
```

Close the Thomson Reuters Elektron connection.

```
close(c)
```

- “Decide to Buy Shares Using Thomson Reuters Elektron Current Data” on page 6-2
- “Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data” on page 6-5

Input Arguments

c — Thomson Reuters Elektron connection
`elektron` object

Thomson Reuters Elektron connection, specified as an `elektron` object created using the `elektron` function.

See Also

See Also
`elektron` | `getdata` | `realtime`

Topics
“Decide to Buy Shares Using Thomson Reuters Elektron Current Data” on page 6-2
“Decide to Buy Shares Using Thomson Reuters Elektron Real-Time Data” on page 6-5

External Websites
Thomson Reuters Elektron

Introduced in R2017a

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

See Also

close | fetch | get

Introduced in R2009a

close

Close Thomson Reuters Tick History connection

Syntax

```
close(r)
```

Description

`close(r)` closes the Thomson Reuters Tick History connection, `r`.

See Also

See Also

`rdth`

Introduced in R2009a

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 `reqtype` and message types `messtype`. The value of `messtype` is a cell array of character vectors. Data for the fields specified by `tradefields` is returned for the data range bounded by `daterange`. The value of `tradefields` is a cell array of multiple cell arrays of character vectors. Each cell array of character vectors in `tradefields` denotes a unique field list for each message type. Specifying the `exchange` of the given security improves the speed of the data request. `domain` specifies the security type.

`x =`
`fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain,marketdepth` additionally specifies the depth of level 2 data, `marketdepth`, to return for a `'MarketDepth'` request type. `marketdepth` must be a numeric value between `1` and `10`, returning up to 10 bid/ask values for a given security.

Note: Do not use date ranges for end of day requests. You can specify a range of hours on a single day, but not a multiple day range.

Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```
returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get information pertaining to a particular security, the command

```
d = fetch(r,'GOOG.O',{'Volume','Price','Exchange ID'},...
{'09/05/2008 12:00:00','09/05/2008 12:01:00'},...
'TimeAndSales','Trade','NSQ','EQU')
```
returns data starting with (not all data is shown):

```
d =
'#RIC'      'Date[L]'        'Time[L]'         'Type'...
     'Ex/Cntrb.ID'    'Price'
'GOOG.O'    '05-SEP-2008'    '12:00:01.178'    'Trade'...
    'NAS'            '443.86'
'Volume'
'200'
```
The command

```
d = fetch(r,'GOOG.O',{'Volume','Last'},{'09/05/2008'},...
'EndOfDay','End Of Day','NSQ','EQU')
```
returns

```
d =
    '#RIC'      'Date[L]'        'Time[L]'       ...
  'Type'    'Last'     'Volume'
    'GOOG.O'    '05-SEP-2008'    '23:59:00.000'   ...
 'End Of Day'    '444.25'    '4538375'
```
For

```
x = fetch(r,'GOOG.O')
```
for example, the exchange of the security is `x.Exchange` or `NSQ`. To determine the asset domain of the security, use the value of `x.Type`, in this case 113. Using the information from `v = get(r)`,

```
j = find(v.InstrumentTypes.Value == 113)
```
returns

```
j =46
```
The command

```
v.InstrumentTypes.Value(j)
```
returns

```
ans =
   113
```
The command

```
v.InstrumentTypes.Name(j)
```
returns

```
ans =
    'Equities'
```
The command

```
v.AssetDomains.Value(strcmp(v.InstrumentTypes.Name(j),...
v.AssetDomains.Name))
```
returns

```
ans =
    'EQU'
```
Knowing the security exchange and domain helps the interface to resolve the security symbol and return data more quickly.

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = fetch(R,'AAPL.O',{'Bid Price','Bid Size'},...
        {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

Tips

- To obtain more information request and message types and their associated field lists, use the command `get(r)`.

See Also

See Also

`close` | `get` | `rdth`

Introduced in R2009a

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 character vector or cell array of character vectors 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

See Also

`fetch` | `rdth`

Introduced in R2009a

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

See Also

`close` | `fetch` | `get` | `rdth`

Introduced in R2009a

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

See Also

`close` | `rdth` | `submitftp`

Introduced in R2011a

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

See Also

`fetch` | `get` | `rdth` | `rdthloader` | `status`

Introduced in R2011a

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 character vectors.
`'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

See Also

`reuters` | `rnseloader`

Introduced in R2008b

reuters

Create Reuters sessions

Before using `reuters`, configure your environment for connecting to a Reuters data server. For details, see “Configuring Reuters Market Data System Connections” on page 1-9.

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)
```

- "Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Input Arguments

`session` — Session name
character vector

Session name, specified as a character vector to denote a Reuters session.

Data Types: `char`

`service` — Service name
character vector

Service name, specified as a character vector to denote the service for connecting to the Reuters data server.

Data Types: `char`

`username` — User name
character vector

User name, specified as a character vector to denote your Reuters user identification.

Data Types: `char`

`ipaddress` — IP address
character vector

IP address, specified as a character vector 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

Tips

- You can connect to the Reuters data server without DACS authentication. For example, use this code.

  ```
  c = reuters('myNS::remoteSession','IDN_CONFLATED');
  ```
- When you connect to 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
```

See Also

See Also

`addric` | `close` | `contrib` | `deleteric` | `fetch` | `get` | `history` | `rmdsconfig` | `stop`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

"Configuring Reuters Market Data System Connections" on page 1-9

Introduced in R2008a

addric

Create Reuters Instrument Code

Syntax

```
addric(c,ric,fid,fval,type)
```

Description

`addric(c,ric,fid,fval,type)` creates a Reuters Instrument Code, `ric`, on the service defined by the Reuters session, `c`. Supply the field ID or name, `fid`, and the field value, `fval`. Specify whether the RIC `type` is `'live'` or `'static'` (default).

Examples

Create a live RIC called `myric` with the fields `'trdprc_1'` (field ID 6) and `'bid'` (field ID 22) set to initial values of `0`:

```
addric(c,'myric',{trdprc_1','bid'},{0,0},'live')
```

Create a live RIC called `myric` with the fields `trdprc_1` and `bid` set to initial values of `0`:

```
addric(c,'myric',{6,22},{0,0},'live')
```

See Also

See Also

`contrib` | `deleteric` | `fetch` | `reuters`

Topics

“Retrieve Current and Historical Data Using Thomson Reuters” on page 1-22

Introduced in R2010b

close

Release connections to Reuters data servers

Syntax

```
close(c)
```

Arguments

`c`	Reuters session object created with `reuters`.

Description

`close(c)` releases the Reuters connection `c`.

Examples

Release the connection `c` to the Reuters data server, and unsubscribe all requests associated with it:

```
close(c)
```

See Also

See Also

`reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2008a

contrib

Contribute data to Reuters data feed

Syntax

```
contrib(c,s,fid,fval)
```

Description

`contrib(c,s,fid,fval)` contributes data to a Reuters data feed. `c` is the Reuters session object, and `s` is the RIC. Supply the field IDs or names, `fid`, and field values, `fval`.

Examples

Contribute data to the Reuters datafeed for the Reuters session object `c` and the RIC `'myric'`. Provide a last trade price of 33.5.

```
contrib(c,'myric','trdprc_1',33.5)
```

Contribute an additional bid price of 33.8:

```
contrib(c,'myric',{'trdprc_1','bid'},{33.5,33.8})
```

Submit value 33.5 for field 6 (`'trdprc_1'`):

```
contrib(c,'myric',6,33.5)
```

Add the value 33.8 to field 22 (`'bid'`):

```
contrib(c,'myric',{6,22},{33.5,33.8})
```

See Also

See Also

`addric` | `deleteric` | `fetch` | `reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2010b

deleteric

Delete Reuters Instrument Code

Syntax

```
deleteric(c,ric)
deleteric(c,ric,fid)
```

Description

`deleteric(c,ric)` deletes the Reuters Instrument Code, `ric`, and all associated fields. `c` is the Reuters session object.

`deleteric(c,ric,fid)` deletes the fields specified by `fid` for the `ric`.

Examples

Delete `'myric'` and all of its fields:

```
deleteric(c,'myric')
```

Delete the fields `'fid1'` and `'fid2'` from `'myric'`:

```
deleteric(c,'myric',{'fid1','fid2'})
```

See Also

See Also

`addric` | `contrib` | `fetch` | `reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2010b

fetch

Request data from Reuters data servers

Syntax

```
d = fetch(c,sec)
d = fetch(c,sec,[],fields)
subs = fetch(c,sec,eventhandler)
```

Description

`d = fetch(c,sec)` returns the current data for the security `sec`, given the Reuters session object `c`.

`d = fetch(c,sec,[],fields)` requests the given fields `fields`, for the security `sec`, given the Reuters session object `c`.

`subs = fetch(c,sec,eventhandler)` uses the Reuters session object `c` to subscribe to the security `sec`. MATLAB runs the `eventhandler` function for each data event that occurs.

Examples

Retrieve Current Securities Data

Connect to 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.

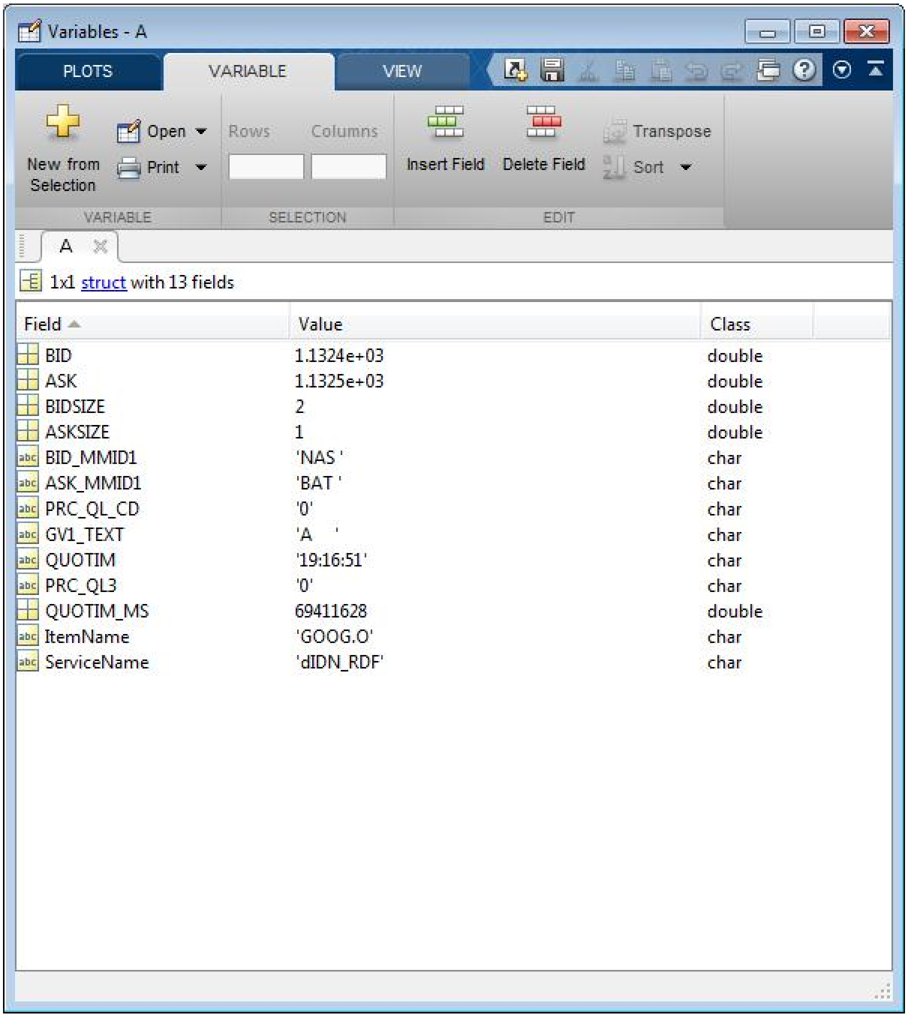
Connect to Thomson Reuters.

```
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')
```



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)
```

- "Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Input Arguments

`c` — Reuters session

object

Reuters session, specified as a Reuters session object created using `reuters`.

`sec` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector or a cell array of character vectors to denote Reuters securities.

Data Types: `char` | `cell`

`fields` — Reuters fields list

character vector | cell array of character vectors

Reuters fields list, specified as a character vector or cell array of character vectors 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.

See Also

See Also

`close` | `reuters` | `stop`

Topics

“Retrieve Current and Historical Data Using Thomson Reuters” on page 1-22
“Writing and Running Custom Event Handler Functions” on page 1-35

Introduced in R2008a

get

Retrieve properties of Reuters session objects

Syntax

```
e = get(c)
e = get(c,f)
```

Arguments

`c`	Reuters session object created with `reuters`.
`f`	Reuters session properties list.

Description

`e = get(c)` returns Reuters session properties for the Reuters session object `c`.

`e = get(c,f)` returns Reuters session properties specified by the properties list `f` for the Reuters session object `c`.

See Also

See Also

`reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2008a

history

Request data from Reuters Time Series One

Syntax

```
d = history(c,s)
d = history(c,s,p)
d = history(c,s,f)
d = history(c,s,f,p)
d = history(c,s,d)
d = history(c,s,startdate,enddate)
d = history(c,s,startdate,enddate,p)
d = history(c,s,f,startdate,enddate)
d = history(c,s,f,startdate,enddate,p)
```

Description

`d = history(c,s)` returns all available daily historical data for the RIC, `s`, for the Reuters session object `c`.

`d = history(c,s,p)` returns all available historical data for the RIC, `s`, for the Reuters session object `c`. `p` specifies the period of the data:

- `'d'` - daily (default)
- `'w'` - weekly
- `'m'` - monthly

Note: Reuters Time Series One will only return two years of daily data, five years of weekly data, or ten years of monthly data from the current date.

`d = history(c,s,f)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`.

`d = history(c,s,f,p)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `c`. `p` specifies the period of the data.

`d = history(c,s,d)` returns the historical data for the RIC, `s`, for the given date, `d`, for the Reuters session object `c`.

`d = history(c,s,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,startdate,enddate,p)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

`d = history(c,s,f,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(c,s,f,startdate,enddate,p)` returns the historical data for the RIC, `s`, and fields, `f`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

Examples

`d = history(c,'WXYZ.O')` returns a structure containing all available historical end of day daily data for the RIC `'WXYZ.O'`, for the Reuters session object `c`.

`d = history(c,'WXYZ.O','close')` returns a structure with the fields `date` and `close` containing all available historical end of day daily data for the RIC `'WXYZ.O'`.

`d = history(c,'WXYZ.O','close','m')` returns all available monthly data.

`d = history(c,'WXYZ.O','01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009. Note that only two years worth of daily data, five years worth of weekly data, and 10 years of monthly data from today's date is made available by Reuters.

`d = history(c,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

`d = history(c,'WXYZ.O',`
`{'close','volume'},'01-03-2009','02-24-2009','w')` returns all available weekly data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

See Also

See Also

`close` | `fetch` | `reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2009b

stop

Unsubscribe securities

Syntax

```
stop(c)
stop(c,d)
```

Arguments

`c`	Reuters session object created with `reuters`.
`d`	Subscription handle returned by `fetch`.

Description

`stop(c)` unsubscribes all securities associated with the Reuters session object `c`.

`stop(c,d)` unsubscribes the securities associated with the subscription handle `d`, where `d` is the subscription handle returned by `reuters/fetch`.

Examples

Unsubscribe securities associated with a specific request `d` and a Reuters connection object `c`:

```
stop(c,d)
```

Unsubscribe all securities associated with the Reuters connection object `c`:

```
stop(c)
```

See Also

See Also

`fetch` | `reuters`

Topics

"Retrieve Current and Historical Data Using Thomson Reuters" on page 1-22

Introduced in R2008a

rmdsconfig

Reuters Market Data System (RMDS) configuration editor

Syntax

```
rmdsconfig
```

Description

`rmdsconfig` opens the Reuters Market Data System configuration editor.

See Also

See Also

`reuters`

Introduced in R2010b

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 character vectors.
`'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

See Also

`rdthloader` | `reuters`

Introduced in R2008b

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`

Introduced in R2011b

close

Close connection to SIX Financial Information

Syntax

```
close(C)
```

Description

`close(C)` closes the connection, `C`, to SIX Financial Information.

See Also

`tlkrs`

Introduced in R2011b

getdata

Current SIX Financial Information data

Syntax

```
D = getdata(c,s,f)
```

Description

`D = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`.

Examples

Retrieve SIX Financial Information pricing data for specified securities.

```
% Connect to Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert specified fields to ID strings.
ids =  tkfieldtoid(c,{'Bid','Ask','Last'},'market');

% Retrieve data for specified securities.
d = getdata(c,{'1758999,149,134','275027,148,184'},ids);
```

Your output appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, and `d.P` contains the pricing data.

View the instrument IDs like this:

```
d.I.k
```

```
ans =
    '1758999,149,134'
    '275027,148,184'
```

View the pricing data field IDs like this:

```
d.P.k

ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
    '33,3,1'
    '3,1,1'
```

And the pricing data like this:

```
d.P.v

ans =

    '44.94'
    '44.95'
    []
    '0.9715'
    '0.9717'
    []
```

Convert field IDs in `d.P.k` to field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'market')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names (`Bid`, `Ask`, `Last`) and corresponding IDs.

See Also

`tlkrs` | `timeseries` | `tkidtofield` | `history` | `tkfieldtoid`

Introduced in R2011b

history

End of day SIX Financial Information data

Syntax

```
D = history(c,s,f,fromdate,todate)
```

Description

`D = history(c,s,f,fromdate,todate)` returns the historical data for the security list `s`, for the fields `f`, for the dates `fromdate` to `todate`.

Examples

Retrieve end of day SIX Financial Information data for the specified security for the past 5 days.

```
c = tlkrs('US12345','userapid01','userapid10')
ids = tkfieldtoid(c,{'Bid','Ask'},'history');
d = history(c,{'1758999,149,134'},ids,floor(now)-5,floor(now));
```

```
d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
     HL: [1x1 struct]
     HD: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.HD` contains the dates, and `d.P` contains the pricing data.

View the dates:

```
d.HD.d

ans =
```

```
    '20110225'
    '20110228'
    '20110301'
```

View the pricing field IDs:

```
d.P.k

ans =

    '3,2'
    '3,3'
    '3,2'
    '3,3'
    '3,2'
    '3,3'
```

View the pricing data:

```
d.P.v

ans =

    '45.32'
    '45.33'
    '45.26'
    '45.27'
    '44.94'
    '44.95'
```

Convert the field identification values in `d.P.k` to their corresponding field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

See Also

`tlkrs` | `timeseries` | `tkidtofield` | `getdata` | `tkfieldtoid`

Introduced in R2011b

isconnection

Determine if SIX Financial Information connection is valid

Syntax

```
X = isconnection(C)
```

Description

`X = isconnection(C)` returns `true` if `C` is a valid SIX Financial Information connection and `false` otherwise.

See Also

`tlkrs` | `close` | `getdata`

Introduced in R2011b

timeseries

SIX Financial Information intraday tick data

Syntax

```
D = timeseries(c,s,t)
D = timeseries(c,s,{startdate,enddate})
D = timeseries(c,s,t,5)
```

Description

`D = timeseries(c,s,t)` returns the raw tick data for the SIX Financial Information connection object `c`, the security `s`, and the date `t`. Every trade, best, and ask tick is returned for the given date or date range.

`D = timeseries(c,s,{startdate,enddate})` returns the raw tick data for the security `s`, for the date range defined by `startdate` and `enddate`.

`D = timeseries(c,s,t,5)` returns the tick data for the security `s`, for the date `t` in intervals of 5 minutes, for the field `f`. Intraday tick data requested is returned in 5-minute intervals, with the columns representing:

- First
- High
- Low
- Last
- Volume weighted average
- Moving average

Examples

Retrieve SIX Financial Information intraday tick data for the past 2 days:

```
c = tlkrs('US12345','userapid01','userapid10')
```

```
d = timeseries(c,{'1758999,149,134'}, ...
   {floor(now)-.25,floor(now)})
```

Display the returned data:

```
d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
    TSL: [1x1 struct]
     TS: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.TS` contains the date and time data, and `d.P` contains the pricing data.

Display the tick times:

```
d.TS.t(1:10)

ans =

    '013500'
    '013505'
    '013510'
    '013520'
    '013530'
    '013540'
    '013550'
    '013600'
    '013610'
    '013620'
```

Display the field IDs:

```
d.P.k(1:10)

ans =

    '3,4'
    '3,2'
    '3,3'
    '3,4'
    '3,2'
```

```
    '3,3'
    '3,4'
    '3,2'
    '3,3'
    '3,4'
```

Convert these IDs to field names (`Mid`, `Bid`, `Ask`) with `tkidtofield`:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

Load the file `@tlkrs/tkfields.mat` for a listing of the field names and corresponding IDs.

Display the corresponding tick values:

```
d.P.v(1:10)

ans =

    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
    '45.32'
    '45.33'
    '45.325'
```

See Also

`tlkrs` | `history` | `getdata`

Introduced in R2011b

tkfieldtoid

SIX Financial Information field names to identification string

Syntax

```
D = tkfieldtoid(c,f,typ)
```

Description

`D = tkfieldtoid(c,f,typ)` converts SIX Financial Information field names to their corresponding identification strings. `c` is the SIX Financial Information connection object, `f` is the field list, and `typ` denotes the field. Options for the field include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with `history`.

Examples

Retrieve pricing data associated with specified identification strings:

```
% Connect to SIX Telekurs.
c = tlkrs('US12345','userapid01','userapid10')

% Convert field names to identification strings.
ids = tkfieldtoid(c,{'bid','ask','last'},'market');

% Retrieve data associated with the identification strings.
d = getdata(c,{'1758999,149,134','275027,148,184',ids);
```

See Also

`tlkrs` | `history` | `tkidtofield` | `getdata` | `timeseries`

Introduced in R2011b

tkidtofield

SIX Financial Information identification string to field name

Syntax

```
D = tkidtofield(c,f,typ)
```

Description

`D = tkidtofield(c,f,typ)` converts SIX Financial Information field identification strings to their corresponding field names. `c` is the SIX Financial Information connection object, `f` is the ID list, and `typ` denotes the fields. Options for the fields include market, `'market'`; time and sales, `'tass'`; and history, `'history'`. `market` fields are used with `getdata`, `tass` fields are used with `timeseries`, and `history` fields are used with the `history`.

Examples

When you retrieve output from SIX Financial Information, it appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

The instrument IDs are found in `d.I`, and the pricing data is found in `d.P`. The output for `d.P.k` appears like this:

```
ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
```

```
'33,3,1'
'3,1,1'
```

Convert the field IDs in `d.P.k` to their field names with `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`

Introduced in R2011b

statsllc

STATS.com connection

Description

The `statsllc` function creates a `statsllc` object. The `statsllc` object represents a STATS.com connection.

You can retrieve athlete, team, and event data using the `statsllc` object. You can retrieve data for the sport and time period that you have access to using your credentials. The credentials consist of an API key and secret pass code. To retrieve data for different athletes, teams, events, and time periods, specify query parameters. For credentials, contact STATS.com. For all available query parameters, see STATS Developer Center I/O Docs.

Create Object

Syntax

Description

`c = statsllc(apikey,secret)` creates an empty STATS.com connection `c` using the API key and secret pass code. To retrieve data with a custom URL, use this syntax with `fetchUrl`. Or, to retrieve data without a custom URL, use this syntax, set the object properties, and run `fetch`.

`c = statsllc(apikey,secret,`
`QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` creates a STATS.com connection `c` using web service query parameters, as specified by one or more name-value pair arguments. The STATS.com web service defines the query parameters. For query parameters, see STATS Developer Center I/O Docs. To retrieve data, use this syntax with `fetch`.

Input Arguments

`apikey` — STATS.com API key
character vector

STATS.com API key, specified as a character vector. To request your API key, contact STATS.com.

Data Types: `char`

`secret` — STATS.com secret pass code
character vector

STATS.com secret pass code, specified as a character vector. To request your secret pass code, contact STATS.com.

Data Types: `char`

`QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters
name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector that specifies the name of a query parameter. A `QueryValue` argument is a character vector that specifies the value of the query parameter.

The web service defines name-value pairs that it accepts as part of a request. For valid name-value pairs, see STATS Developer Center I/O Docs. To understand which name-value pairs you have access to, check your license or contact STATS.com.

Example:
`statsllc('gkfrq6fabcfehmn2yctrc6j5','aBC5cuBQgc','DataType','stats','LeagueAbb`
retrieves the standings for the current or most recent baseball season.

Data Types: `char`

Properties

`DataType` — STATS.com type of data
character vector

STATS.com type of data, specified as a character vector.

The type `'stats'` means statistical data. The type `'optical'` means spatial data that is derived from video recordings. For other types of data, contact STATS.com.

Data Types: `char`

`LeagueAbbreviation` — Abbreviation for a sports league name
character vector

Abbreviation for a sports league name, specified as a character vector.

Example: `'mlb'`

Data Types: `char`

`Method` — Player or team identifier
character vector

Player or team identifier, specified as a character vector.

Example: `'548033'`

Data Types: `char`

`Resource` — Type of information to request
character vector

Type of information to request, specified as a character vector.

To request data for games, specify `'events'`. To request statistical data for a player, specify `'stats/players'`. For other resources, contact STATS.com.

Data Types: `char`

`SportName` — Sport name
character vector

Sport name, specified as a character vector.

Example: `'baseball'`

Data Types: `char`

`URL` — URL
`'http://api.stats.com'` (default) | character vector

URL, specified as a character vector. STATS.com uses a custom-built URL to make the data request. For details about the URL, contact STATS.com.

The base URL stub for creating the web request is `'http://api.stats.com'`.

Data Types: `char`

`VersionNumber` — STATS.com version number
character vector

STATS.com version number, specified as a character vector. For details about the STATS.com version number, contact STATS.com.

Example: `'v1'`

Data Types: `char`

Object Functions

fetch	Retrieve data from STATS.com
fetchUrl	Retrieve data from STATS.com with URL

Examples

Connect to STATS.com

Connect to STATS.com using an API key and secret pass code.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret)

sBasketball =

  statsllc with properties:

              DataType: []
    LeagueAbbreviation: []
                Method: []
              Resource: []
             SportName: []
```

```
              URL: 'http://api.stats.com'
    VersionNumber: []
```

`sBasketball` is the STATS.com connection object.

Retrieve basketball statistics for individual players in the sport league named `'nba'`. To specify the statistics for retrieval, create a URL suffix.

```
urlsuffix = '/v1/stats/basketball/nba/participants';
```

```
d = fetchUrl(sBasketball,urlsuffix);
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetchUrl`.

Retrieving data from STATS.com indicates a successful connection. If there are connection issues, an error displays. For details about errors, see "STATS.com Data Retrieval Errors" on page 4-2.

Connect to STATS.com by Setting Object Properties

Connect to STATS.com using an API key and secret pass code.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';
```

```
sBasketball = statsllc(apikey,secret)
```

```
sBasketball = 

  statsllc with properties:

             DataType: []
   LeagueAbbreviation: []
               Method: []
             Resource: []
            SportName: []
                  URL: 'http://api.stats.com'
        VersionNumber: []
```

`sBasketball` is the STATS.com connection object.

Set the object properties in `sBasketball` to create a specific data request. Specify statistical standings data for basketball teams in the sport league named `'nba'`.

```
sBasketball.DataType = 'stats';
sBasketball.LeagueAbbreviation = 'nba';
sBasketball.Resource = 'standings';
sBasketball.SportName = 'basketball';
sBasketball.VersionNumber = 'v1';
```

Retrieve basketball team data using the STATS.com connection `sBasketball`.

```
d = fetch(sBasketball);
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetch`.

Retrieving data from STATS.com indicates a successful connection. If there are connection issues, an error displays. For details about errors, see "STATS.com Data Retrieval Errors" on page 4-2.

Connect to STATS.com with Query Parameters

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1')

sBasketball = 

  statsllc with properties:

              DataType: []
    LeagueAbbreviation: []
                Method: []
              Resource: []
             SportName: []
                   URL: 'http://api.stats.com'
         VersionNumber: []
```

`sBasketball` is the STATS.com connection object.

Retrieve the basketball team data for the 2015 season.

```
d = fetch(sBasketball,'season','2015');
```

`d` contains statistical basketball data for individual players. For details about data retrieval, see `fetch`.

Retrieving data from STATS.com indicates a successful connection. If there are connection issues, an error displays. For details about errors, see “STATS.com Data Retrieval Errors” on page 4-2.

- “Compare Player Salaries by Injury Status” on page 3-27
- “Retrieve Team Standings for the Current Year” on page 3-32

See Also

Topics

“Compare Player Salaries by Injury Status” on page 3-27
“Retrieve Team Standings for the Current Year” on page 3-32
“STATS.com Data Retrieval Errors” on page 4-2

External Websites

STATS.com
STATS Developer Center I/O Docs

Introduced in R2016b

fetch

Retrieve data from STATS.com

Syntax

```
d = fetch(c)
d = fetch(c,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)
```

Description

`d = fetch(c)` returns STATS.com data using the STATS.com connection `c`.

`d = fetch(c,QueryName1,QueryValue1,...,QueryNameN,QueryValueN)` returns STATS.com data using web service query parameters, as specified by one or more name-value pair arguments. The STATS.com web service defines the query parameters. For query parameters, see STATS Developer Center I/O Docs.

Examples

Retrieve STATS.com Data

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1');
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball team data using the STATS.com connection `sBasketball`.

```
d = fetch(sBasketball)
```

```
d = 

  struct with fields:

            status: 'OK'
       recordCount: 30
    startTimestamp: '2016-04-19T18:37:36.0745302Z'
      endTimestamp: '2016-04-19T18:37:36.9026552Z'
         timeTaken: 0.8281
        apiResults: [1×1 struct]
```

The query returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request
- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve the basketball team data `f` for the second conference in the structure array.

```
f = d.apiResults.league.season.eventType.conferences(2)
```

```
f = 

  struct with fields:

    conferenceId: 2
            name: 'Western Conference'
    abbreviation: 'Western'
       divisions: [3×1 struct]
```

`f` is a structure with these fields:

- `conferenceId` — Conference identifier
- `name` — Conference name
- `abbreviation` — Conference abbreviation

- `divisions` — Divisions in the conference

Retrieve the team identifiers in the first division in the structure array `divisions`.

```
f.divisions(1).teams.teamId

ans =

     9

ans =

    12
...
```

Retrieve STATS.com Data Using Query Parameters

Connect to STATS.com using an API key and secret pass code. Specify statistical standings data for basketball teams in the sport league named `'nba'` using query parameters.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret, ...
    'DataType','stats','LeagueAbbreviation','nba', ...
    'Resource','standings','SportName','basketball', ...
    'VersionNumber','v1');
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball team data using the STATS.com connection `sBasketball` for the 2015 season.

```
d = fetch(sBasketball,'season','2015')

d =

  struct with fields:

            status: 'OK'
       recordCount: 30
    startTimestamp: '2016-04-19T18:37:36.0745302Z'
```

```
      endTimestamp: '2016-04-19T18:37:36.9026552Z'
         timeTaken: 0.8281
        apiResults: [1×1 struct]
```

The query returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request
- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve the basketball team data `f` for the second conference in the structure array.

```
f = d.apiResults.league.season.eventType.conferences(2)
```

```
f = 

  struct with fields:

    conferenceId: 2
            name: 'Western Conference'
    abbreviation: 'Western'
       divisions: [3×1 struct]
```

`f` is a structure with these fields:

- `conferenceId` — Conference identifier
- `name` — Conference name
- `abbreviation` — Conference abbreviation
- `divisions` — Divisions in the conference

Retrieve the team identifiers in the first division in the structure array `divisions`.

```
f.divisions(1).teams.teamId
```

```
ans =

     9
```

```
ans =

    12
...
```

- "Compare Player Salaries by Injury Status" on page 3-27
- "Retrieve Team Standings for the Current Year" on page 3-32

Input Arguments

c — STATS.com connection
`statsllc` object

STATS.com connection, specified as a `statsllc` object.

`QueryName1,QueryValue1,...,QueryNameN,QueryValueN` — Web service query parameters
name-value pairs

Web service query parameters, specified as one or more pairs of name-value arguments. A `QueryName` argument is a character vector that specifies the name of a query parameter. A `QueryValue` argument is a character vector that specifies the value of the query parameter.

The web service defines name-value pairs that it accepts as part of a request. For valid name-value pairs, see STATS Developer Center I/O Docs. To understand which name-value pairs you have access to, contact STATS.com.

The name-value pairs for this function are different from the name-value pairs specified in `statsllc`. For details about the differences, contact STATS.com.

Example: `'season','2015'` retrieves data for the 2015 season.

Data Types: `char`

Output Arguments

d — STATS.com data
structure

STATS.com data, returned as a structure with these fields:

STATS.com Data Structure Field	Description
`status`	Web request status (`'OK'` denotes a successful web request)
`recordCount`	Number of records returned
`startTimestamp`	Start date and time of the web request
`endTimestamp`	End date and time of the web request
`timeTaken`	Number of seconds to complete the web request
`apiResults`	Returned data

To retrieve individual or team athletic data, access the field `apiResults` in the structure. For example, enter:

```
d.apiResults
```

```
ans =

  struct with fields:

    sportId: 1
       name: 'Basketball'
     league: [1×1 struct]
```

For more data, continue to access the structure fields that contain further data. For accessing nested structures, see “Access Data in Nested Structures” (MATLAB).

See Also

See Also
`fetchUrl` | `statsllc`

Topics
“Compare Player Salaries by Injury Status” on page 3-27
“Retrieve Team Standings for the Current Year” on page 3-32
“STATS.com Data Retrieval Errors” on page 4-2

External Websites

STATS.com
STATS Developer Center I/O Docs

Introduced in R2016b

fetchUrl

Retrieve data from STATS.com with URL

Syntax

```
d = fetchUrl(c,urlsuffix)
```

Description

`d = fetchUrl(c,urlsuffix)` returns STATS.com data using the STATS.com connection `c` and a custom URL suffix.

Examples

Retrieve STATS.com Data Using Custom URL

Connect to STATS.com using an API key and secret pass code.

```
apikey = 'gkfrq6fabcfehmn2yctrc6j5';
secret = 'aBC5cuBQgc';

sBasketball = statsllc(apikey,secret);
```

`sBasketball` is the STATS.com connection object. For each object property, see `statsllc`.

Retrieve basketball statistics for individual players in the sport league named `'nba'`. To specify the statistics for retrieval, create a URL suffix.

```
urlsuffix = '/v1/stats/basketball/nba/participants';

d = fetchUrl(sBasketball,urlsuffix)

d =

  struct with fields:
```

```
            status: 'OK'
       recordCount: 450
    startTimestamp: '2016-04-26T15:28:46.2203857Z'
      endTimestamp: '2016-04-26T15:28:46.892265Z'
         timeTaken: 0.6719
        apiResults: [1×1 struct]
```

STATS.com returns a structure `d` with these fields:

- `status` — Web request status (`'OK'` denotes a successful web request)
- `recordCount` — Number of records returned
- `startTimestamp` — Start date and time of the web request
- `endTimestamp` — End date and time of the web request
- `timeTaken` — Amount of time taken for the web request to complete in seconds
- `apiResults` — Returned data

Retrieve data about the first player in the league from the returned data `d.apiResults`. Access the nested structure to retrieve data.

```
d.apiResults.league.players{1}
```

```
ans = 

  struct with fields:

    isSuspended: 0
      isInjured: 1
     highSchool: [1×1 struct]
...
```

The structure result contains suspended status, injury status, and high school information along with other fields. For details about these fields, contact STATS.com.

For more data, continue to access the nested structures or structure arrays. For accessing nested structures, see “Access Data in Nested Structures” (MATLAB).

Input Arguments

c — STATS.com connection
`statsllc` object

STATS.com connection, specified as a `statsllc` object.

`urlsuffix` — URL suffix
character vector

URL suffix, specified as a character vector. `fetchUrl` builds a custom URL by appending the URL suffix to the connection URL created using `statsllc`. The `fetchUrl` function uses the custom URL to retrieve STATS.com data. To compose the URL suffix, specify query parameter values. For query parameter values, see STATS Developer Center I/O Docs.

Example: `'/v1/stats/baseball/mlb/participants'` retrieves statistical data for individual baseball players in the sports league named `'mlb'`

Data Types: `char`

Output Arguments

`d` — STATS.com data
structure

STATS.com data, returned as a structure with these fields:

STATS.com Data Structure Field	Description
`status`	Web request status (`'OK'` denotes a successful web request)
`recordCount`	Number of records returned
`startTimestamp`	Start date and time of the web request
`endTimestamp`	End date and time of the web request
`timeTaken`	Number of seconds to complete the web request
`apiResults`	Returned data

To retrieve individual or team athletic data, access the field `apiResults` in the structure. For example, enter:

```
d.apiResults
```

```
ans =
```

```
  struct with fields:

    sportId: 1
       name: 'Basketball'
     league: [1×1 struct]
```

For more data, continue to access the structure fields that contain further data. For accessing nested structures, see "Access Data in Nested Structures" (MATLAB).

See Also

See Also

`fetch` | `statsllc`

Topics

"STATS.com Data Retrieval Errors" on page 4-2

External Websites

STATS.com
STATS Developer Center I/O Docs

Introduced in R2016b

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)
```

- "Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Output Arguments

c — Yahoo! connection

connection object

Yahoo! connection, returned as a connection object.

See Also

See Also

`builduniverse` | `close` | `fetch` | `get` | `isconnection` | `trpdata`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced before R2006a

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)
```

- "Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Input Arguments

`c` — Yahoo! connection

connection object

Yahoo! connection, specified as a connection object created using `yahoo`.

`s` — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors for multiple securities. These character vectors must be in a format recognizable by the Yahoo! server.

Data Types: `char` | `cell`

`fromdate` — Beginning date

numeric scalar | vector | matrix | character vector | cell array of character vectors

Beginning date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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

numeric scalar | vector | matrix | character vector | cell array of character vectors

End date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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 values.

Value	Description
`'d'`	Daily
`'w'`	Weekly
`'m'`	Monthly

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.

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

See Also

`fetch` | `trpdata`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced in R2011b

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

See Also
`yahoo`

Topics
“Retrieve Current and Historical Data Using Yahoo!” on page 1-33

Introduced before R2006a

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)
```

- “Retrieve Current and Historical Data Using Yahoo!” on page 1-33

Input Arguments

c — Yahoo! connection

connection object

Yahoo! connection, specified as a connection object created using `yahoo`.

s — Security list

character vector | cell array of character vectors

Security list, specified as a character vector for one security or a cell array of character vectors for more than one security. These character vectors 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

character vector | serial date number

Request date, specified as a character vector 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
numeric scalar | vector | matrix | character vector | cell array of character vectors

Beginning date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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
numeric scalar | vector | matrix | character vector | cell array of character vectors

End date for the historical data, specified as a double scalar, double vector, double matrix, character vector, or cell array of character vectors. 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'` | `'w'` | `'m'` | `'v'`

Period within a date range, specified as one of these values:

- `'d'` for daily
- `'w'` for weekly
- `'m'` for monthly
- `'v'` for dividends

`f` — Request fields
character vector | cell array of character vectors

Request fields, specified as a character vector or cell array of character vectors 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

See Also

`close` | `get` | `isconnection` | `yahoo`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced before R2006a

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 character vector or cell array of character vectors 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

See Also

`close` | `fetch` | `isconnection` | `yahoo`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced before R2006a

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

See Also

`close` | `fetch` | `get` | `yahoo`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced before R2006a

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 character vector, `d1` is the start date, `d2` is the end date, and `p` is the periodicity flag for Yahoo!, generates a total return price series. `prc` is the price, `act` is the action, and `div` is the dividend returned in the total return price series.

Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by the `trpdata` function contains the expected results.

See Also

See Also

`close` | `yahoo`

Topics

"Retrieve Current and Historical Data Using Yahoo!" on page 1-33

Introduced in R2011b