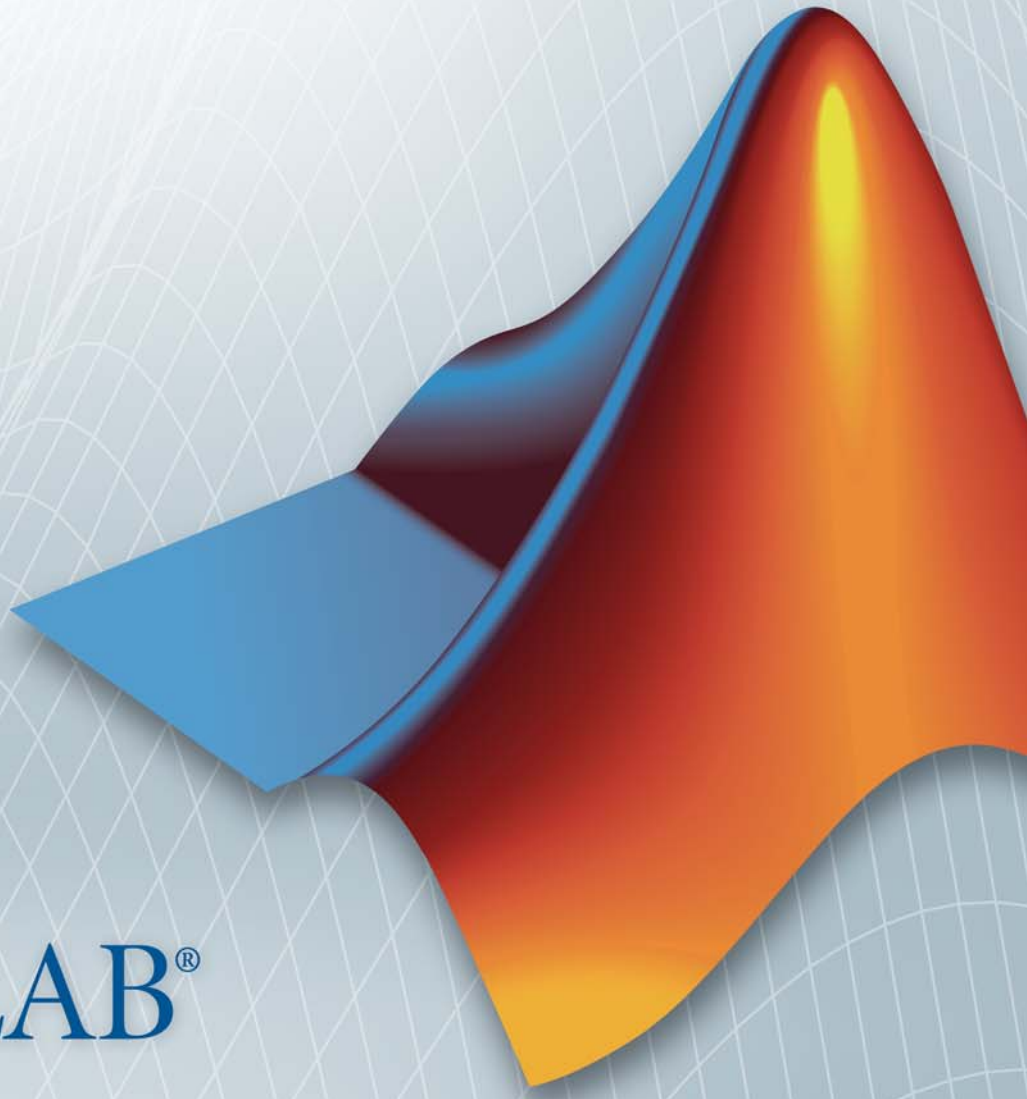
Datafeed Toolbox™

User's Guide

R2012a



MATLAB®

MathWorks®

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Datafeed Toolbox™ User's Guide

© COPYRIGHT 1999–2012 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Revision History

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

Contents

1 Getting Started

Product Description 1-2
Key Features 1-2

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

2 Communicate with Financial Data Servers

Communicate with Data Servers 2-2

Connect to the Bloomberg Data Server 2-3

Connection Object Properties 2-4
Retrieve Connection Properties 2-4
Example: Retrieve Data on a Security 2-5

Disconnect from Data Servers 2-6

X_TRADER Price Update 2-7

X_TRADER Price Update Depth 2-9

X_TRADER Order Submission 2-13

3 Example: Retrieve Bloomberg Data

About This Example 3-2

Retrieve Field Data 3-3

Retrieve Time-Series Data 3-4

Retrieve Historical Data 3-5

4 Datafeed Toolbox Graphical User Interface

Introduction 4-2

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

Obtain Ticker Symbol with the Datafeed Securities Lookup Dialog Box 4-9

5 Function Reference

Bloomberg 5-2

Datastream 5-3

eSignal 5-4

FactSet 5-5

FRED 5-6

Haver Analytics 5-7

Interactive Data Pricing and RemotePlus 5-8

Kx Systems 5-9

Reuters 5-10

SIX Telekurs 5-11

Thomson Reuters Tick History 5-12

Reuters Newscope 5-13

Yahoo! 5-14

Trading Technologies 5-15

6 Functions — Alphabetical List

A Examples

Retrieving Connection Properties A-2

Index

1

Getting Started

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

Product Description

Access financial data from data service providers

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

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

Key Features

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

About Data Servers and Data Service Providers

In this section...

"Supported Data Service Providers" on page 1-3

"Data Server Connection Requirements" on page 1-4

Supported Data Service Providers

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

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

  Note Only Bloomberg Desktop API is supported.

- eSignal® (`http://www.esignal.com`)
- FactSet® Research Systems, Inc. (`http://www.factset.com`)
- Federal Reserve Economic Data (FRED) (`http://research.stlouisfed.org/fred2/`)
- Haver Analytics (`http://www.haver.com`)
- Interactive Data™ (`http://www.interactivedata-prd.com/`)
- Kx Systems®, Inc. (`http://www.kx.com`)
- SIX Telekurs™ (`http://www.telekurs-financial.com/`)
- Thomson Reuters (`http://www.thomsonreuters.com/`)
- Trading Technologies® X_TRADER® (`http://www.tradingtechnologies.com`)
- Yahoo!® (`http://finance.yahoo.com`)

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

Data Server Connection Requirements

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

Additional Software Requirements

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

- Bloomberg

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

- Interactive Data Pricing and Reference Data's RemotePlus™
- Haver Analytics
- Kx Systems, Inc.
- Reuters®

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

```
Invalid MEX-file
```

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

Proxy Information Requirements

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

- FactSet
- Federal Reserve Economic Data
- Thomson Reuters Datastream®

- Thomson Reuters Tick History
- Yahoo!

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

FactSet Data Service Requirements

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

Reuters Data Service Requirements

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

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

   ```
   rmdsconfig
   ```

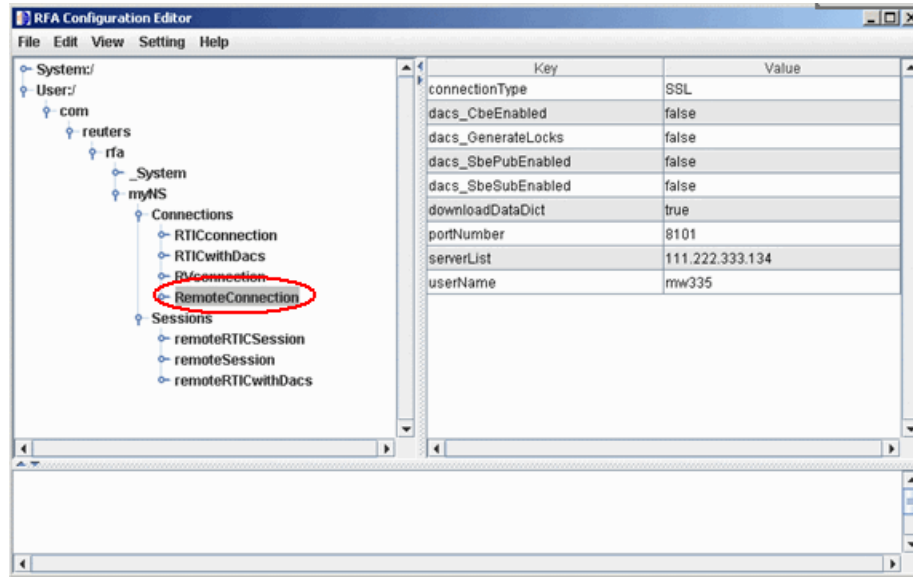
2. Load the sample configuration file.

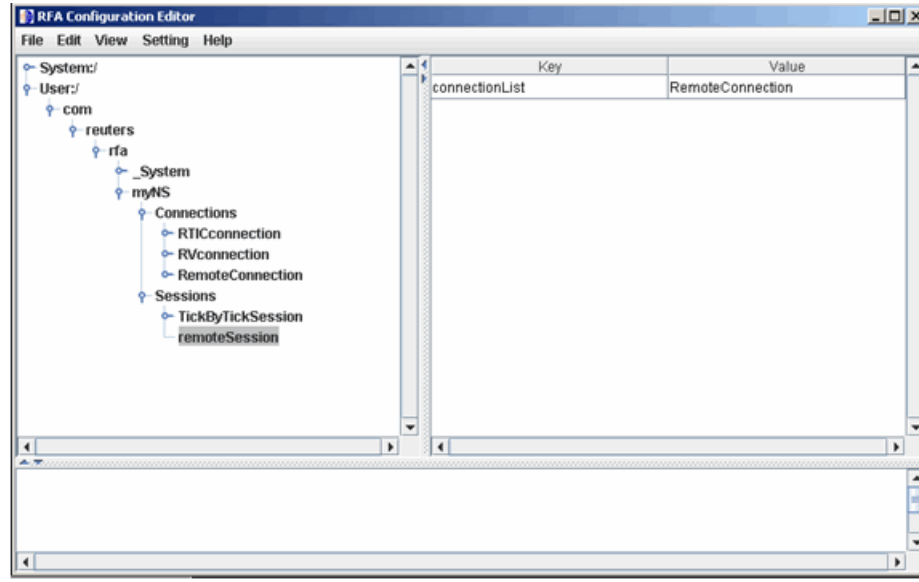
   a. Click **File > Import > File**.

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

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

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





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

a Add the following to your connection configuration:

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

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

```
dacs_GenerateLocks=false
```

For more information, see the `reuters` function reference page.

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

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

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

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

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

Thomson Reuters Data Service Requirements

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

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

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

2

Communicate with Financial Data Servers

- "Communicate with Data Servers" on page 2-2
- "Connect to the Bloomberg Data Server" on page 2-3
- "Connection Object Properties" on page 2-4
- "Disconnect from Data Servers" on page 2-6
- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9
- "X_TRADER Order Submission" on page 2-13

Communicate with Data Servers

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

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

Data Server	Toolbox Function
Bloomberg	`blp`
eSignal	`esig`
FactSet	`factset`
FRED	`fred`
Haver Analytics	`haver`
Interactive Data	`idc`
Kx Systems, Inc.	`kx`
SIX Telekurs	`tlkrs`
Thomson Reuters	`datastream` or `reuters`
Trading Technologies X_TRADER	`xtrdr`
Yahoo!	`yahoo`

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

Connect to the Bloomberg Data Server

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

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

2. Enter the following command:

   ```
   c = blp
   ```

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

   ```
   c =

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

Connection Object Properties

In this section...

"Retrieve Connection Properties" on page 2-4

"Example: Retrieve Data on a Security" on page 2-5

The syntax for the Bloomberg V3 connection object constructor is:

```
c = blp;
```

Retrieve Connection Properties

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

```
get(c)

c =

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

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

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

ans =

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

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

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

Note A single property is not returned as a structure.

Example: Retrieve Data on a Security

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

```
c = blp;
```

Use the `blp.timeseries` method to return data on a security:

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

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

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

Disconnect from Data Servers

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

```
close(c)
```

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

X_TRADER Price Update

This example shows how to connect to X_TRADER and listen for price update event data.

Connect to X_TRADER.

Start or connect to X_TRADER.

```
X = xtrdr;
```

Create an event notifier.

```
createNotifier(X);
```

The event notifier is the X_TRADER mechanism that lets you define MATLAB functions to use as callbacks for specific events.

Create an instrument.

Create an instrument, and attach it to the notifier.

```
createInstrument(X,'Exchange','CME','Product','ES',...
                'ProdType','Future','Contract','Dec11',...
                'Alias','PriceInstrument1');
X.InstrNotify(1).AttachInstrument(X.Instrument(1));
```

Define events.

Assign callbacks for validating or invalidating an instrument, and for handling data updates for a previously validated instrument.

```
X.InstrNotify(1).registerevent({'OnNotifyFound',...
                      @(varargin)ttinstrumentfound(varargin{:})});
X.InstrNotify(1).registerevent({'OnNotifyNotFound',...
                      @(varargin)ttinstrumentnotfound(varargin{:})});
X.InstrNotify(1).registerevent({'OnNotifyUpdate',...
                      @(varargin)ttinstrumentupdate(varargin{:})});
```

Monitor events.

Set the update filter to monitor only the desired fields. Here, monitor events where the last price, last quantity, the previous last quantity, and the change in prices are updated. Listen for event data.

```
X.InstrNotify(1).UpdateFilter = 'Last$,LastQty$,~LastQty$,Change$';
X.Instrument(1).Open(0);
```

The last command tells X_TRADER to start monitoring the attached instruments using the specified event settings.

Close the connection.

Close the X_TRADER connection.

```
close(X)
```

See Also

`xtrdr | xtrdr.close | xtrdr.createInstrument | xtrdr.createNotifier`

Related Examples

- “X_TRADER Price Update Depth” on page 2-9
- “X_TRADER Order Submission” on page 2-13

X_TRADER Price Update Depth

This example shows how to connect to X_TRADER and turn on event handling for level two market that (for example, bid and ask orders in the market for an instrument). Create a figure window to display the depth data.

Connect to X_TRADER.

Start or connect to X_TRADER.

```
X = xtrdr;
```

Create an event notifier.

Create an event notifier and enable depth updates.

```
createNotifier(X);
X.InstrNotify(1).EnableDepthUpdates = 1;
```

The event notifier is the X_TRADER mechanism that lets you define MATLAB functions to use as callbacks for specific events.

Create an instrument.

```
createInstrument(X,'Exhange','CME','Product','ES','ProdType','Future',...
                'Contract','Dec11','Alias','PriceInstrumentDepthUpdate');
```

Attach an instrument to a notifier.

```
X.InstrNotify(1).AttachInstrument(X.Instrument(1));
```

You can assign one or more instruments to a notifier, and a notifier can have one or more instruments attached to it.

Define events.

Assign callbacks for validating or invalidating an instrument, and updating the example order book window.

```
X.InstrNotify(1).registerevent({'OnNotifyFound',...
                                    @ttinstrumentfound});
X.InstrNotify(1).registerevent({'OnNotifyNotFound',...
```

```
                                    @ttinstrumentnotfound});
X.InstrNotify(1).registerevent({'OnNotifyDepthData',...
                                    @ttinstrumentdepthupdate});
```

Set up the figure window.

Set up the figure window to display depth data.

```
figure('Numbertitle','off','Tag','TTPriceUpdateDepthFigure',...
       'Name',['Order Book - ' X.Instrument(1).Alias]);
pos = get(gcf,'Position');
set(gcf,'Position',[pos(1) pos(2) 360 315],'Resize','off');
```

Create controls.

Create controls for the last price data.

```
bspc = 5;
bwid = 80;
bhgt = 20;

uicontrol('Style','text','String','Exchange',...
          'Position',[bspc 4*bspc+3*bhgt bwid bhgt]);
uicontrol('Style','text','String','Product',...
          'Position',[2*bspc+bwid 4*bspc+3*bhgt bwid bhgt]);
uicontrol('Style','text','String','Type',...
          'Position',[3*bspc+2*bwid 4*bspc+3*bhgt bwid bhgt]);
uicontrol('Style','text','String','Contract',...
          'Position',[4*bspc+3*bwid 4*bspc+3*bhgt bwid bhgt]);
ui.Exchange = uicontrol('Style','text','Tag','',...
          'Position',[bspc 3*bspc+2*bhgt bwid bhgt]);
ui.Product = uicontrol('Style','text','Tag','',...
          'Position',[2*bspc+bwid 3*bspc+2*bhgt bwid bhgt]);
ui.Type = uicontrol('Style','text','Tag','',...
          'Position',[3*bspc+2*bwid 3*bspc+2*bhgt bwid bhgt]);
ui.Contract = uicontrol('Style','text','Tag','',...
          'Position',[4*bspc+3*bwid 3*bspc+2*bhgt bwid bhgt]);
uicontrol('Style','text','String','Last Price',...
          'Position',[bspc 2*bspc+bhgt bwid bhgt]);
uicontrol('Style','text','String','Last Qty',...
          'Position',[2*bspc+bwid 2*bspc+bhgt bwid bhgt]);
```

```
uicontrol('Style','text','String','Change',...
          'Position',[3*bspc+2*bwid 2*bspc+bhgt bwid bhgt]);
ui.Last = uicontrol('Style','text','Tag','',...
          'Position',[bspc bspc bwid bhgt]);
ui.Quantity = uicontrol('Style','text','Tag','',...
          'Position',[2*bspc+bwid bspc bwid bhgt]);
ui.Change = uicontrol('Style','text','Tag','',...
          'Position',[3*bspc+2*bwid bspc bwid bhgt]);
```

Create a table.

Create a table containing order information.

```
data = {' '};
data = data(ones(10,4));
uibook = uitable('Data',data,'ColumnName',...
                {'Bid','Bid Size','Ask','Ask Size'},...
                 'Position',[5 105 350 205]);
```

Store data.

```
setappdata(0,'TTOrderBookHandle',uibook)
setappdata(0,'TTOrderBookUIData',ui)
```

Listen for event data.

Listen for event data with depth updates enabled.

```
X.Instrument(1).Open(1);
```

The last command tells X_TRADER to start monitoring the attached instruments using the specified event settings.

Close the connection.

Close the X_TRADER connection.

```
close(X)
```

See Also

`xtrdr` | `xtrdr.close` | `xtrdr.createInstrument` | `xtrdr.createNotifier` | `xtrdr.getData`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Order Submission" on page 2-13

X_TRADER Order Submission

This example shows how to connect to X_TRADER and submit an order.

Connect to X_TRADER.

Start or connect to X_TRADER.

```
X = xtrdr;
```

Create an instrument.

```
createInstrument(X,'Exchange','CME','Product','ES',...
                'ProdType','Future','Contract','Dec11',...
                'Alias','SubmitOrderInstrument1');
```

Register event handlers.

Register event handlers for the order server. The callback `ttorderserverstatus` is assigned to the event `OnExchangeStateUpdate` in order to verify that the requested instrument's exchange order server is running. Otherwise, no orders can be submitted.

```
sExchange = X.Instrument.Exchange;
X.Gate.registerevent({'OnExchangeStateUpdate',...
      @(varargin)ttorderserverstatus(varargin{:},sExchange)});
```

Create an order set.

The `OrderSet` object is used to send orders to X_TRADER.

Set properties of the `OrderSet` object and detail the level of the order status events. Enable order update and reject (failure) events so you can assign callbacks to handle these conditions.

```
createOrderSet(X);
X.OrderSet(1).EnableOrderRejectData = 1;
X.OrderSet(1).EnableOrderUpdateData = 1;
X.OrderSet(1).OrderStatusNotifyMode = 'ORD_NOTIFY_NORMAL';
```

Set position limit checks.

Set whether the order set checks self-imposed position limits when submitting an order.

```
X.OrderSet(1).Set('NetLimits',false);
```

Set a callback function.

Set a callback to handle the `OnOrderFilled` events. Each time an order is filled (or partially filled), this callback is invoked.

```
X.OrderSet(1).registerevent({'OnOrderFilled',...
                    @(varargin)ttorderevent(varargin{:},X)});
```

Enable send orders.

Enable send orders. This lets you submit orders to X_TRADER.

```
X.OrderSet(1).Open(1);
```

Build an order profile.

Build an order profile using an existing instrument. The order profile contains the settings that define a submitted order. The valid `Set` parameters are shown in the following examples.

```
orderProfile = createOrderProfile(X);
orderProfile.Instrument = X.Instrument(1);
orderProfile.Customer = '<Default>';
```

Example: Create a market order.

Create a market order to buy 100 shares.

```
orderProfile.Set('BuySell','Buy');
orderProfile.Set('Qty',100);
orderProfile.Set('OrderType','M');
```

Example: Create a limit order.

Create a limit order by setting the `OrderType` and limit order price.

```
orderProfile.Set('OrderType','L');
```

```
orderProfile.Set('Limit$','127000');
```

Example: Create a stop market order.

Create a stop market order, and set the order restriction to stop order and stop price.

```
orderProfile.Set('OrderType','M');
orderProfile.Set('OrderRestr','S');
orderProfile.Set('Stop$','129800');
```

Example: Create a stop limit order.

Create a stop limit order, and set order restriction, type, limit price, and stop price.

```
orderProfile.Set('OrderType','L');
orderProfile.Set('OrderRestr','S');
orderProfile.Set('Limit$','128000');
orderProfile.Set('Stop$','1287500');
```

Check order server status.

Check the order server status before submitting the order, and add a counter so the example never gets stuck.

```
nCounter = 1;
while ~exist('bServerUp','var') && nCounter < 20
   pause(1)
   nCounter = nCounter + 1;
end
```

Verify order server availability.

Verify that the exchange's order server in question is available before submitting the order.

```
if exist('bServerUp','var') && bServerUp
   submittedQuantity = X.OrderSet(1).SendOrder(orderProfile);
   disp(['Quantity Sent: ' num2str(submittedQuantity)])
else
   disp('Order Server is down. Unable to submit order')
```

```
end
```

Close the connection.

Close the X_TRADER connection.

```
close(X)
```

See Also

`xtrdr | xtrdr.close | xtrdr.createInstrument | xtrdr.createOrderProfile | xtrdr.createOrderSet`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9

3

Example: Retrieve Bloomberg Data

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

About This Example

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

Note If you have not used the `blp` function before you will need to add the file `blpapi3.jar` to the MATLAB Java classpath. Use the `javaaddpath` function or edit your `classpath.txt` file.

Retrieve Field Data

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

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

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

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

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

Retrieve Time-Series Data

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

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

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

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

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

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

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

Retrieve Historical Data

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

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

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

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

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

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

4

Datafeed Toolbox Graphical User Interface

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

Introduction

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

This GUI consists of two dialog boxes:

- The Datafeed dialog box
- The Securities Lookup dialog box

Retrieve Data with the Datafeed Dialog Box

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

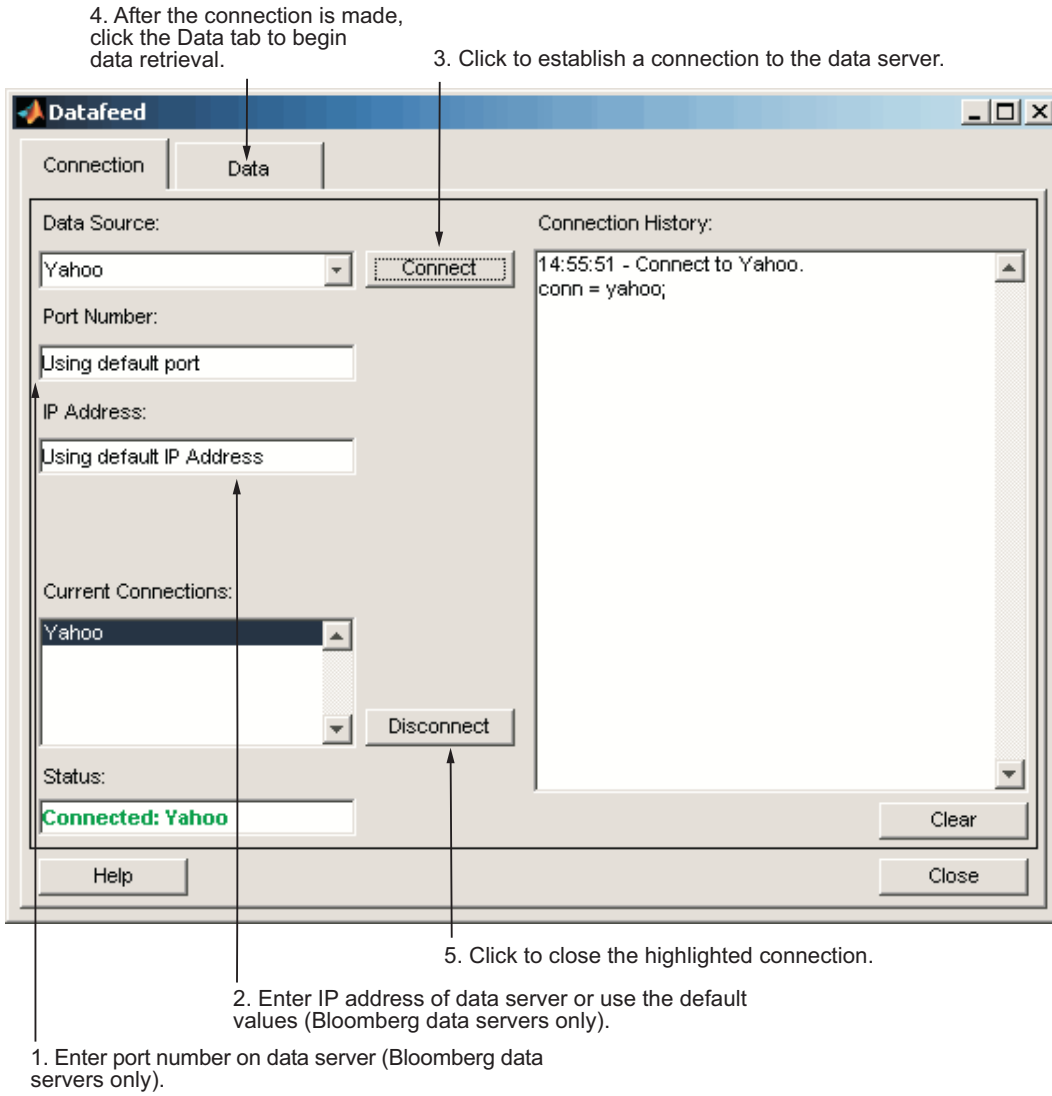
- Bloomberg
- Interactive Data Pricing and Reference Data's RemotePlus
- Yahoo!

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

The Datafeed dialog box consists of two tabs:

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

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



The Datafeed Dialog Box

Connecting to Data Servers

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

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

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

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

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

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

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

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

Retrieving Data

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

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

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

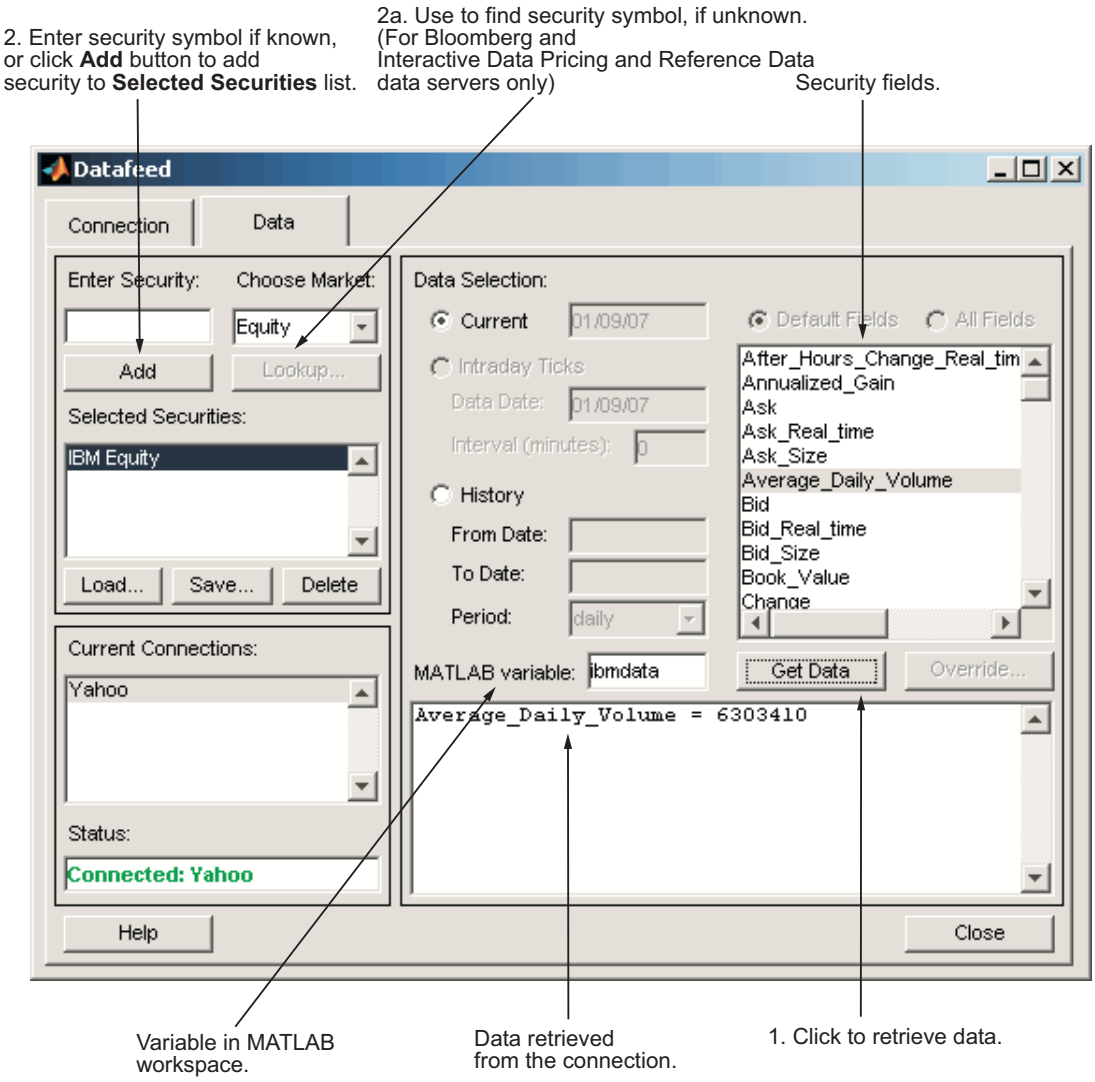
3 Specify whether you want the default set of data, or the full set:

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

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

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

The following figure summarizes these steps.



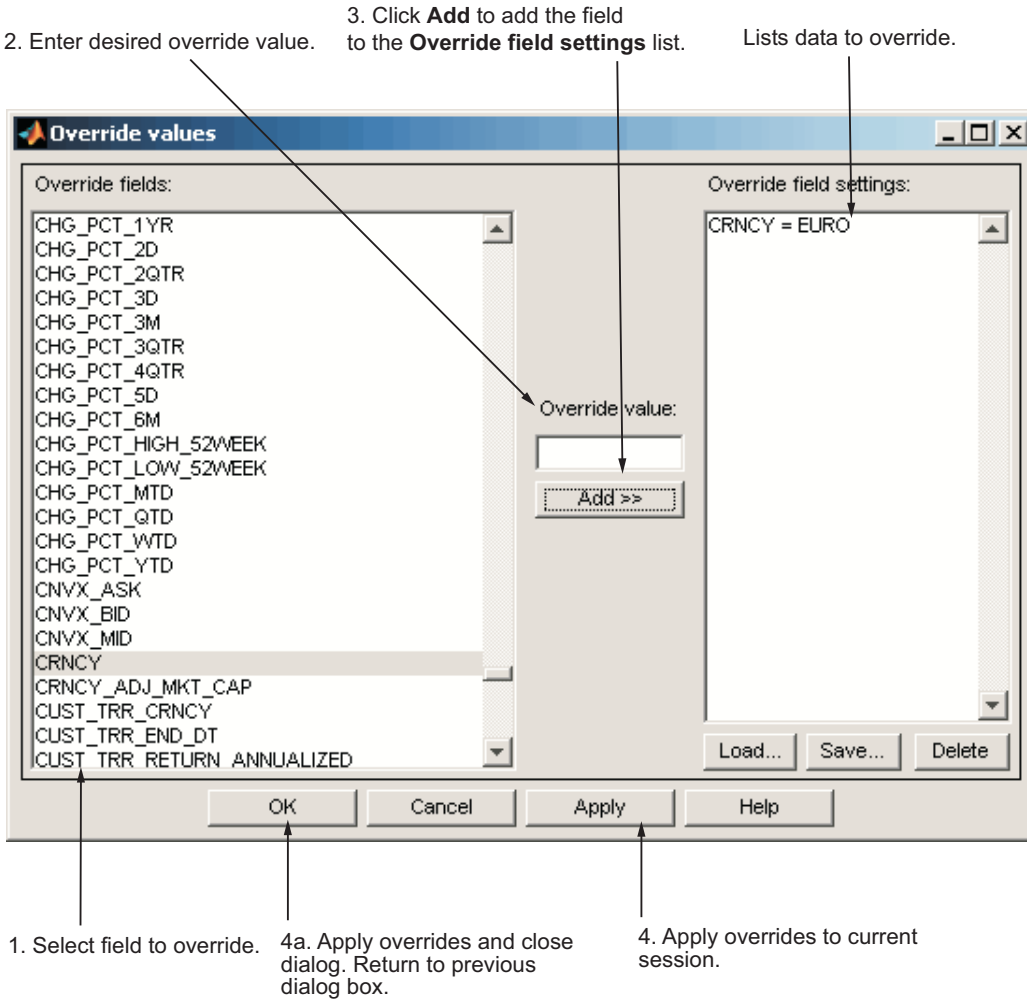
Setting Overrides

To set overrides on retrieved data:

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

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

The following figure summarizes these steps.


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

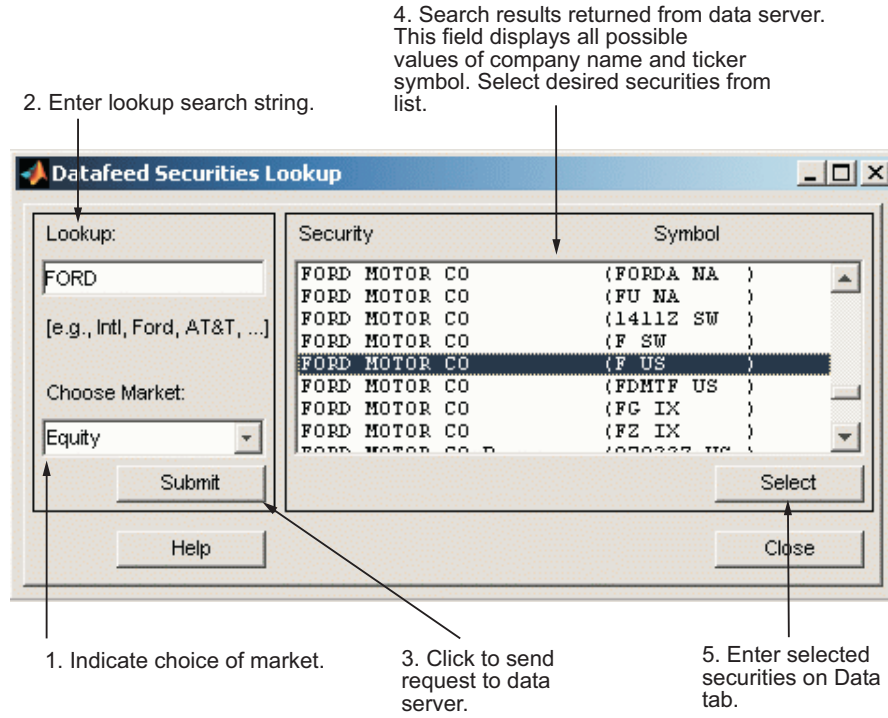
Obtain Ticker Symbol with the Datafeed Securities Lookup Dialog Box

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

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

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

The following figure summarizes these steps.


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

5

Function Reference

Bloomberg (p. 5-2)	Get Bloomberg financial data
Datastream (p. 5-3)	Get Thomson Reuters Datastream financial data
eSignal (p. 5-4)	Get eSignal financial data
FactSet (p. 5-5)	Get FactSet financial data
FRED (p. 5-6)	Get Federal Reserve Economic Data (FRED) financial data
Haver Analytics (p. 5-7)	Get Haver Analytics financial data
Interactive Data Pricing and RemotePlus (p. 5-8)	Get Interactive™ Data Pricing and Reference Data's RemotePlus™ financial data
Kx Systems (p. 5-9)	Get Kx Systems, Inc. kdb+ financial data
Reuters (p. 5-10)	Get Reuters financial data
SIX Telekurs (p. 5-11)	Get SIX Telekurs financial data
Thomson Reuters Tick History (p. 5-12)	Retrieve data from Thomson Reuters Tick History file
Reuters Newscope (p. 5-13)	Retrieve data from Reuters Newscope sentiment archive file
Yahoo! (p. 5-14)	Get Yahoo! financial data
Trading Technologies (p. 5-15)	X_TRADER market data and order management

Bloomberg

`blp`	Bloomberg V3 communications server connection
`blp.category`	Bloomberg V3 field category search
`blp.close`	Close connection to Bloomberg V3 data server
`blp.display`	Display Bloomberg V3 connection object
`blp.fieldinfo`	Bloomberg V3 field information
`blp.fieldsearch`	Bloomberg V3 field search
`blp.get`	Get Bloomberg V3 connection properties
`blp.getdata`	Current Bloomberg V3 data
`blp.history`	Bloomberg V3 historical data
`blp.isconnection`	`true` if valid Bloomberg V3 connection
`blp.realtime`	Bloomberg V3 real-time data retrieval
`blp.stop`	Unsubscribe real time requests for Bloomberg V3
`blp.timeseries`	Bloomberg V3 intraday tick data

Datastream

`datastream`	Establish connections to Thomson Reuters Datastream API
`datastream.close`	Close connections to Thomson Reuters Datastream data servers
`datastream.fetch`	Request data from Thomson Reuters Datastream data servers
`datastream.get`	Retrieve properties of Thomson Reuters Datastream connection objects
`datastream.isconnection`	Verify whether connections to Thomson Reuters Datastream data servers are valid

eSignal

`esig`	eSignal Desktop API connection
`esig.close`	Close eSignal connection
`esig.getdata`	Current eSignal data
`esig.getfundamentaldata`	Current eSignal fundamenal data
`esig.history`	eSignal historical data
`esig.timeseries`	eSignal intraday tick data

FactSet

`factset`	Establish connections to FactSet data servers
`factset.close`	Close connections to FactSet data servers
`factset.fetch`	Request data from FactSet data servers
`factset.get`	Retrieve properties of FactSet connection objects
`factset.isconnection`	Verify whether connections to FactSet data servers are valid

FRED

`fred`	Connect to FRED data servers
`fred.close`	Close connections to FRED data servers
`fred.fetch`	Request data from FRED data servers
`fred.get`	Retrieve properties of FRED connection objects
`fred.isconnection`	Verify whether connections to FRED data servers are valid

Haver Analytics

`haver`	Connect to local Haver Analytics database
`haver.aggregation`	Set Haver Analytics aggregation mode
`haver.close`	Close Haver Analytics database
`haver.fetch`	Request data from Haver Analytics database
`haver.get`	Retrieve properties from Haver Analytics connection objects
`haver.info`	Retrieve information about Haver Analytics variables
`haver.isconnection`	Verify whether connections to Haver Analytics data servers are valid
`haver.nextinfo`	Retrieve information about next Haver Analytics variable
`havertool`	Run Haver Analytics graphical user interface (GUI)

Interactive Data Pricing and RemotePlus

`idc`	Connect to Interactive Data Pricing and Reference Data's RemotePlus data servers
`idc.close`	Close connections to Interactive Data Pricing and Reference Data's RemotePlus data servers
`idc.fetch`	Request data from Interactive Data Pricing and Reference Data's RemotePlus data servers
`idc.get`	Retrieve properties of Interactive Data Pricing and Reference Data's RemotePlus connection objects
`idc.isconnection`	Verify whether connections to Interactive Data Pricing and Reference Data's RemotePlus data servers are valid

Kx Systems

`kx`	Connect to Kx Systems, Inc. kdb+ databases
`kx.close`	Close connections to Kx Systems, Inc. kdb+ databases
`kx.exec`	Run Kx Systems, Inc. kdb+ commands
`kx.fetch`	Request data from Kx Systems, Inc. kdb+ databases
`kx.get`	Retrieve Kx Systems, Inc. kdb+ connection object properties
`kx.insert`	Write data to Kx Systems, Inc. kdb+ databases
`kx.isconnection`	Verify whether connections to Kx Systems, Inc. kdb+ databases are valid
`kx.tables`	Retrieve table names from Kx Systems, Inc. kdb+ databases

Reuters

`reuters`	Create Reuters sessions
`reuters.addric`	Create Reuters Instrument Code
`reuters.close`	Release connections to Reuters data servers
`reuters.contrib`	Contribute data to Reuters datafeed
`reuters.deleteric`	Delete Reuters Instrument Code
`reuters.fetch`	Request data from Reuters data servers
`reuters.get`	Retrieve properties of Reuters session objects
`reuters.history`	Request data from Reuters Time Series One
`reuters.stop`	Unsubscribe securities
`rmdsconfig`	Reuters Market Data System configuration editor

SIX Telekurs

`tlkrs`	SIX Telekurs connection
`tlkrs.close`	Close connection to SIX Telekurs
`tlkrs.getdata`	Current SIX Telekurs data
`tlkrs.history`	End of day SIX Telekurs data
`tlkrs.isconnection`	True if valid SIX Telekurs connection
`tlkrs.timeseries`	SIX Telekurs intraday tick data
`tlkrs.tkfieldtoid`	SIX Telekurs field names to identification string
`tlkrs.tkidtofield`	SIX Telekurs identification string to field name

Thomson Reuters Tick History

`rdth`	Connect to Thomson Reuters Tick History
`rdth.close`	Close Thomson Reuters Tick History connection
`rdth.fetch`	Request Thomson Reuters Tick History data
`rdth.get`	Get Thomson Reuters Tick History connection properties
`rdth.isconnection`	Verify whether Thomson Reuters Tick History connections are valid
`rdth.status`	Status of FTP request for Thomson Reuters Tick History data
`rdth.submitftp`	Submit FTP request for Thomson Reuters Tick History data
`rdthloader`	Retrieve data from Thomson Reuters Tick History file

Reuters Newscope

`rnseloader`	Retrieve data from Reuters Newscope sentiment archive file

Yahoo!

`yahoo`	Connect to Yahoo! data servers
`yahoo.builduniverse`	Portfolio matrix with total return price data from Yahoo!
`yahoo.close`	Close connections to Yahoo! data servers
`yahoo.fetch`	Request data from Yahoo! data servers
`yahoo.get`	Retrieve properties of Yahoo! connection objects
`yahoo.isconnection`	Verify whether connections to Yahoo! data servers are valid
`yahoo.trpdata`	Total return price series data

Trading Technologies

`xtrdr`	X_TRADER connection
`xtrdr.close`	Terminate X_TRADER connection
`xtrdr.createInstrument`	Instruments for X_TRADER
`xtrdr.createNotifier`	Instrument notifier for X_TRADER
`xtrdr.createOrderProfile`	Order profiles for X_TRADER
`xtrdr.createOrderSet`	Order set for X_TRADER
`xtrdr.getData`	Current X_TRADER data

6

Functions — Alphabetical List

bloomberg

Purpose

Connect to Bloomberg data servers

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

Syntax

```
c = bloomberg
```

Description

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

Examples

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

```
c = bloomberg
```

See Also

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

Purpose

Close connections to Bloomberg data servers

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

Syntax

```
close(Connect)
```

Arguments

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

Description

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

Examples

Establish a Bloomberg connection `c`:

```
c = bloomberg
```

Close this connection:

```
close(c)
```

See Also

`bloomberg`

bloomberg.fetch

Purpose

Request data from Bloomberg data servers

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

Syntax

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

Description

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

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

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

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

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

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

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

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

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

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

For today's tick data, enter the command:

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

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

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

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

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

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

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

`data = fetch(Connect, 'Security', 'REALTIME', 'Fields', 'MATLABProg')` subscribes to a given security or list of securities, requesting the indicated fields, and runs any specified MATLAB

function. See `pricevol`, `showtrades`, or `stockticker` for information on the data returned by asynchronous Bloomberg events.

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

Arguments

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

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

Note This argument is case sensitive.

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

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

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

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

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

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

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

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

`Days` Values:

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

`Data` Values:

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

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

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

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

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

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

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

Examples

Retrieving Header Data

Retrieve header data for a United States equity with ticker `ABC`:

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

Retrieving Opening and Closing Prices

Retrieve the opening and closing prices:

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

Retrieving Override Fields

Retrieve the requested fields, given override fields and values:

```
D = fetch(c, '3358ABCD4 Corp', 'GETDATA',...
```

```
{'YLD_YTM_ASK', 'ASK', 'OAS_SPREAD_ASK', 'OAS_VOL_ASK'},...
{'PX_ASK', 'OAS_VOL_ASK'}, {'99.125000', '14.000000'})
```

Retrieving Time Series Data

Retrieve today's time series:

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

Retrieving Time Series Data, Aggregated into Time Intervals

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

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

Retrieving Time Series Default Closing Price

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

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

Retrieving Monthly Closing Price

Retrieve the monthly closing price for the specified dates:

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

See Also

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

bloomberg.get

Purpose

Retrieve Bloomberg connection object properties

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

Syntax

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

Arguments

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

Description

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

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

Examples

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

```
c = bloomberg
```

Retrieve this connection's properties:

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

See Also

bloomberg | bloomberg.close | bloomberg.fetch | bloomberg.isconnection

bloomberg.getdata

Purpose

Current Bloomberg data

Syntax

```
d = getdata(c,s,f)
d = getdata(c,s,f,o,ov)
```

Description

`d = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`. This method uses the Bloomberg ActiveX® interface and does not support overrides. For more robust functionality, refer to the `bloomberg.fetch` method.

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

Examples

The command

```
d = getdata(c,'ABC US Equity',{'LAST_PRICE';'OPEN'})
```

returns the today's current and open price of the given security.

The command

```
d = getdata(c,'3358ABCD4 Corp',...
  {'YLD_YTM_ASK','ASK','OAS_SPREAD_ASK','OAS_VOL_ASK'},...
  {'ASK','OAS_VOL_ASK'},{'99.125000','14.000000'})
```

returns the requested fields given override fields and values.

See Also

`bloomberg.fetch` | `bloomberg.history` | `bloomberg.lookup` | `bloomberg.realtime` | `bloomberg.timeseries`

Purpose

Historical Bloomberg data

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

Syntax

```
d = history(c,s,f,fromdate,todate)
d = history(c,s,f,fromdate,todate,per)
d = history(c,s,f,fromdate,todate,per,cur)
```

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`. This method uses the Bloomberg ActiveX interface. For more robust functionality, refer to the `bloomberg.fetch` method.

`d = history(c,s,f,fromdate,todate,per)` returns the historical data for the field, `f`, for the dates `fromdate` to `todate`. `per` specifies the period of the data:

`'d'`	Daily.
`'w'`	Weekly.
`'m'`	Monthly.
`'q'`	Quarterly.
`'y'`	Yearly.
`' o'`	Omit all days for which there is no data.
`' i'`	Include all trading days.
`' a'`	Include all calendar days.
`'  b'`	Report missing data using Bloomberg default.
`'  s'`	Show missing data as last found value.
`'  n'`	Report missing data as `Nan`.

For example, `per = 'dan'` returns daily data for all calendar days reporting missing data as `NaN`'s. `per = '  n'` returns the data using the default periodicity and default calendar reporting missing data as `NaN`'s.

If you do not specify `per`, the method uses default period for the data.

`d = history(c,s,f,fromdate,todate,per,cur)` returns the historical data for the security list `s` for the fields `f` for the dates `fromdate` to `todate` based on the given currency, `cur`. Load the file `bloomberg/bbfields` to see the list of supported currencies.

Examples

Example 1

The command

```
d = history(c,'ABC US Equity','LAST_PRICE','8/01/99',...
'8/10/99')
```

returns the closing price for the given dates for the given security using the default period of the data.

Example 2

The command

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

returns the monthly closing price for the given dates for the given security.

Example 3

The command

```
D = HISTORY(c,'ABC US Equity','LAST_PRICE','8/01/99',...
'8/10/99','m','USD')
```

returns the monthly closing price converted to US dollars for the given dates for the given security.

Example 4

The command

```
D = HISTORY(c,'ABC US Equity','LAST_PRICE','8/01/99',...
'8/10/99',[],'USD')
```

returns the closing price converted to US dollars for the given dates for the given security using the default period of the data.

bloomberg.isconnection

Purpose

Verify whether connections to Bloomberg data servers are valid

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

Syntax

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

Arguments

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

Description

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

Examples

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

```
c = bloomberg
```

Verify that `c` is a valid connection:

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

See Also

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

Purpose

Verify if valid Bloomberg field

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

Syntax

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

Description

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

Examples

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

returns

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

See Also

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

bloomberg.lookup

Purpose

Bloomberg security search

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

Syntax

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

Description

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

Examples

The command

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

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

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

See Also

`bloomberg.fetch` | `bloomberg.getdata`

Purpose

Bloomberg realtime data retrieval

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

Syntax

```
realtime(c,s,f,api)
```

Description

`realtime(c,s,f,api)` subscribes to a given security or list of securities `s` requesting the fields `f` and runs the specified function by `api`. See the function `bloomberg.showtrades` for information on the data returned by asynchronous Bloomberg events.

Examples

The command

```
realtime(c,'ABC US Equity',{'Last_Trade','Volume'},...
'stockticker')
```

subscribes to the security `ABC US Equity` requesting the fields `Last_Trade` and `Volume` to update in realtime running the function `stockticker`.

See Also

`bloomberg.fetch` | `bloomberg.getdata` | `bloomberg.pricevol` | `bloomberg.stop` | `bloomberg.stockticker` | `bloomberg.showtrades`

bloomberg.pricevol

Purpose

Price and volume (demonstration)

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

Syntax

```
pricevol(InputList)
```

Arguments

`InputList`	Fields from which you request real-time data.

Description

`pricevol(InputList)` demonstrates the Bloomberg real-time data import functionality, where `InputList` is an input list of elements as described in the following table.

`InputList(1) = COM.Bloomberg.Data.1`	Bloomberg handle
`InputList(2) = 1`	Event ID
`InputList(3) = ('Security')`	Security string
`InputList(4) = 1`	Cookie
`InputList(5) = 2`	Field number ID
`InputList(6) = {[43.58]}`	Return data for the given tick
`InputList(7) = 0`	Status
`InputList(8)`	Structure containing the previous fields
`InputList(9) = 'Data'`	Event type

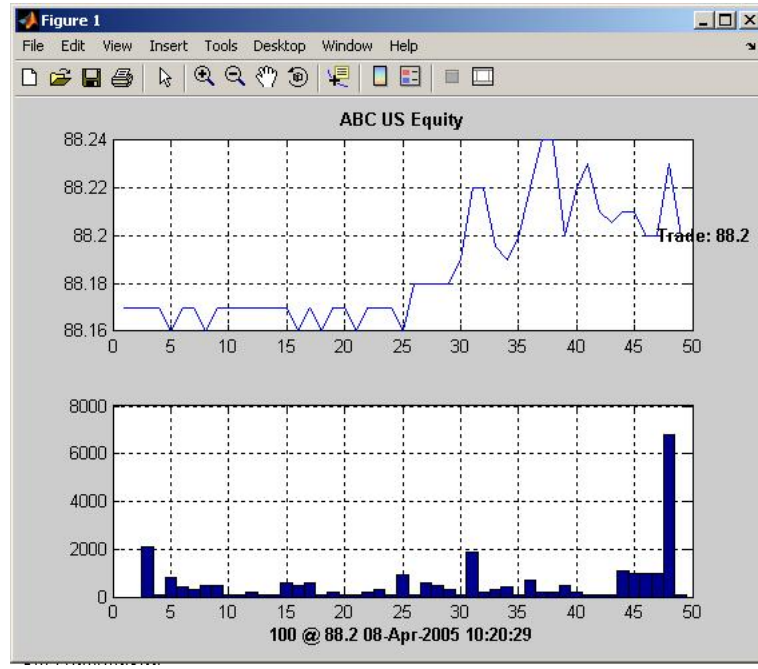
The input argument `InputList(8)` contains the information required to process real-time events.

Examples

Display the most recent `Trade` and `Volume` values in a figure window and show the most recent trade with volumes:

```
c = bloomberg;
d = fetch(c, 'ABC US Equity', 'REALTIME', ...
```

```
{'Last_Trade', 'Volume'}, 'pricevol');
```



See Also

`bloomberg.showtrades` | `bloomberg.stockticker`

bloomberg.showtrades

Purpose

Recent trade data (demonstration)

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

Syntax

```
showtrades(InputList)
```

Arguments

`InputList`	Fields from which you request real-time data.

Description

`showtrades(InputList)` demonstrates the Bloomberg real-time data import functionality, where `InputList` is an input list of elements as described in the following table:

`InputList(1) = COM.Bloomberg.Data.1`	Bloomberg handle
`InputList(2) = 1`	Event ID
`InputList(3) = ('Security')`	Security string
`InputList(4) = 1`	Cookie
`InputList(5) = 2`	Field number ID
`InputList(6) = {[43.58]}`	Return data for the given tick
`InputList(7) = 0`	Status
`InputList(8)`	Structure containing the above fields
`InputList(9) = 'Data'`	Event type

The input argument `InputList(8)` contains the information required to process real-time events.

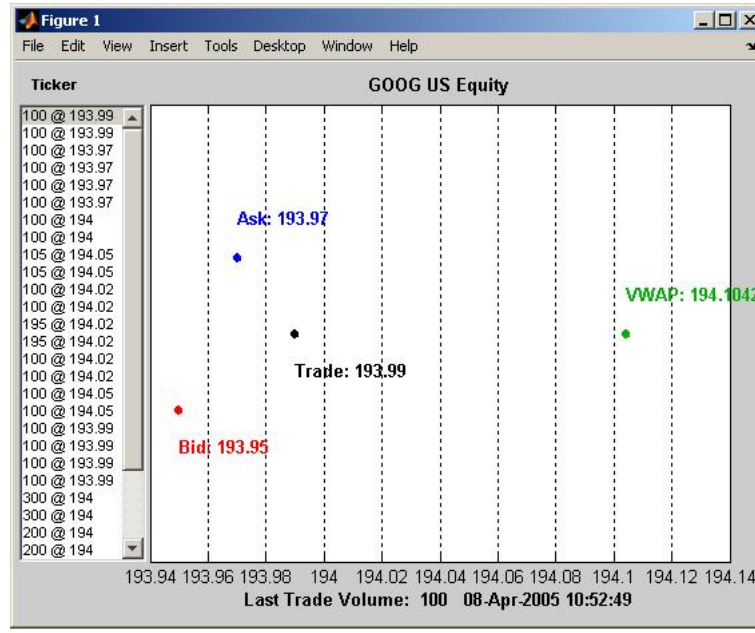
Examples

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

```
c = bloomberg;
```

Display the most recent `Trade`, `Bid`, `Ask`, and `VWAP` (volume-weighted adjusted price), and a list of the most recent trades with volumes:

```
d = fetch(c, 'GOOG US Equity', 'REALTIME', ...
{'Last_Trade','Bid','Ask','Volume','VWAP'},'showtrades');
```



See Also

`bloomberg.pricevol` | `bloomberg.stockticker`

bloomberg.stockticker

Purpose

Trades with volumes (demonstration)

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

Syntax

```
stockticker(InputList)
```

Arguments

`InputList`	Fields from which you request real-time data.

Description

`stockticker(InputList)` demonstrates the Bloomberg real-time data import functionality, where `InputList` is an input list of elements as described in the following table.

`InputList(1) = COM.Bloomberg.Data.1`	Bloomberg handle
`InputList(2) = 1`	Event ID
`InputList(3) = ('Security')`	Security string
`InputList(4) = 1`	Cookie
`InputList(5) = 2`	Field number ID
`InputList(6) = {[43.58]}`	Return data for the given tick
`InputList(7) = 0`	Status
`InputList(8)`	Structure containing the above fields
`InputList(9) = 'Data'`	Event type

The input argument `InputList(8)` contains the information required to process real-time events.

Examples

Retrieve a list of trades with volumes for each requested security:

```
c = bloomberg;
d = fetch(c, {'IBM US Equity', 'EMC US Equity', 'NTAP US Equity'}, ...
'REALTIME', {'Last_Trade', 'Volume'}, 'stockticker');
```

```
** EMC US Equity ** 0 @ 12.65 08-Apr-2005 10:24:57
** IBM US Equity ** 0 @ 88.17 08-Apr-2005 10:24:57
** NTAP US Equity ** 0 @ 29.02 08-Apr-2005 10:24:57
** EMC US Equity ** 200 @ 12.66 08-Apr-2005 10:24:58
** EMC US Equity ** 1400 @ 12.65 08-Apr-2005 10:24:58
** EMC US Equity ** 3100 @ 12.66 08-Apr-2005 10:25:00
** IBM US Equity ** 1300 @ 88.17 08-Apr-2005 10:25:00
.
.
.
```

See Also

bloomberg.pricevol | bloomberg.showtrades

bloomberg.stop

Purpose

Stop Bloomberg real-time data retrieval

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

Syntax

```
stop(c,s)
stop(c,s,api)
```

Description

`stop(c,s)` desubscribes a given security or list of securities `s`. Data events for the security no longer process.

`stop(c,s,api)` desubscribes a given security or list of securities `s` and unregisters the specified callback `api`.

To desubscribe all securities and turn off data event handling, use `close(c)`.

The Bloomberg connection is now closed.

Examples

Example 1

The command

```
stop(c,'ABC US Equity')
```

desubscribes from the security `ABC US Equity`.

Example 2

The command

```
stop(c,'ABC US Equity','stockticker')
```

desubscribes from the security `ABC US Equity` and turns off data event handling by the function `stockticker`.

Example 3

The command

```
STOP(c,'','STOCKTICKER')
```

turns off data event handling by the function `stockticker`.

See Also

bloomberg.fetch | bloomberg.getdata | bloomberg.pricevol | bloomberg.realtime | bloomberg.stockticker | bloomberg.showtrades

bloomberg.timeseries

Purpose

Bloomberg intraday tick data

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

Syntax

```
d = timeseries(c,s,f,t)
d = timeseries(c,s,f,{startdate,enddate})
d = timeseries(c,s,f,t,b)
```

Description

`d = timeseries(c,s,f,t)` returns the tick data for the security `s` for the date `t`. This method uses the Bloomberg ActiveX interface. For more robust functionality, refer to the `bloomberg.fetch` method.

`d = timeseries(c,s,f,{startdate,enddate})` returns the tick data for the security `s` for the date range defined by `startdate` and `enddate`.

`d = timeseries(c,s,f,t,b)` returns the tick data for the security `s` for the date `t` in intervals of `b` minutes for the field, `f`. Intraday tick data requested with an interval is returned with the columns representing `Time`, `Open`, `High`, `Low`, `Last Price` and `Volume` of the ticks in the bar.

Examples

Example 1

The command

```
d = timeseries(c,'ABC US Equity','Last Price',floor(now))
```

returns today's time series for the given security with the timestamp and tick value.

Example 2

The command

```
d = timeseries(c,'ABC US Equity','Last Price',floor(now),5)
```

returns today's `Last Price` series for the given security aggregated into 5-minute intervals minute intervals.

Example 3

The command

```
d = timeseries(c,'ABC US Equity','Last Price',...
{'12/08/2008 00:00:00','12/10/2008 23:59:59.99'},5)
```

returns the `Last Price` series for 12/08/2008 and 12/10/2008 for the given security, aggregated into 5-minute intervals.

See Also

`bloomberg.fetch` | `bloomberg.getdata` | `bloomberg.history`

blp

Purpose

Bloomberg V3 communications server connection

Syntax

```
c = blp
c = blp(P,IP,etimeout)
```

Description

`c = blp` makes a connection to the local Bloomberg V3 communications server. You must have a Bloomberg software license for the host on which the Datafeed Toolbox and MATLAB software are running.

`c = blp(P,IP,etimeout)` makes a connection to the local Bloomberg communications server. `P` is the port number and `IP` is the IP address of the local machine. `etimeout` is the time out value (in milliseconds), specifying how long the connection is attempted before timing out if the connection cannot be made.

Note If you have not used the `blp` function before, you will need to add the file `blpapi3.jar` to the MATLAB Java classpath. Use the `javaaddpath` function or edit your `classpath.txt` file. For more information, see "Using Java Libraries from MATLAB".

Examples

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

```
c = blp
```

Establish a connection using the default port of 8194 and `'localhost'` as the IP address, with a timeout value of 10 seconds.

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

See Also

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

Purpose

Bloomberg V3 field category search

Syntax

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

Description

`d = category(c,f)` returns category information given a search string, `f`. The data returned is an `N`-by-5 cell array containing categories, field IDs, field mnemonics, field names, and field data types for each of the `N` rows in the data set.

See Also

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

blp.close

Purpose

Close connection to Bloomberg V3 data server

Syntax

```
close(c)
```

Description

`close(c)` closes the connection, `c`, to the Bloomberg V3 session.

Purpose

Display Bloomberg V3 connection object

Syntax

```
disp = display(c)
```

Description

`disp = display(c)` displays the Bloomberg V3 connection object.

blp.fieldinfo

Purpose

Bloomberg V3 field information

Syntax

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

Description

`d = fieldinfo(c,f)` returns field information on Bloomberg V3 connection object `c` given a field mnemonic, `f`. The data returned is a `M`-by-5 cell array containing the field help, field ID, field mnemonic, field name, and field data type.

See Also

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

Purpose

Bloomberg V3 field search

Syntax

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

Description

`d = fieldsearch(c,f)` returns field information on Bloomberg V3 connection object `c` given a search string, `f`. The data returned is an `N`-by-5 cell array containing categories, field IDs, field mnemonics, field names, and field data types.

See Also

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

blp.get

Purpose

Get Bloomberg V3 connection properties

Syntax

```
v = get(c,'PropertyName')
v = get(c)
```

Description

`v = get(c,'PropertyName')` returns the value of the specified properties for the Bloomberg V3 connection object. `'PropertyName'` is a string or cell array of strings containing property names. The property names are `session`, `ipaddress`, and `port`.

`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.

See Also

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

Purpose

Current Bloomberg V3 data

Syntax

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

Description

`[d,sec] = getdata(c,s,f)` returns the data for the fields `f` for the security list `s`. `sec` is the security list that maps the order of the return data. The return data, `d` and `sec`, is sorted to match the input order of `s`. You can return securities with any of the following IDs:

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

`[d,sec] = getdata(c,s,f,o,ov)` returns the data for the fields `f` for the security list `s` using the override fields `o` with corresponding override values `ov`.

`[d,sec] = getdata(c,s,f,o,ov,name,value,...)` returns the data for the fields `f` for the security list `s` using the override fields `o` with corresponding override values `ov`. `name`/`value` pairs are used for additional Bloomberg request settings.

Examples

Return today's current and open price of the given security:

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

Return the requested fields given override fields and values:

```
[d,sec] = getdata(c,'3358ABCD4 Corp',{'YLD_YTM_ASK','ASK', ...
   'OAS_SPREAD_ASK','OAS_VOL_ASK'},{'ASK','OAS_VOL_ASK'}, ...
   {'99.125000','14.000000'})
```

See Also

blp | blp.history | blp.realtime | blp.timeseries

Purpose

Bloomberg V3 historical data

Syntax

```
[d, sec] = history(c, s, f, FromDate, ToDate)
[d, sec] = history(c, s, f, FromDate, ToDate, per)
[d, sec] = history(c, s, f, FromDate, ToDate, per, cur)
[d, sec] = history(c, s, f, FromDate, ToDate, per, cur, Name,
   Value, ...)
```

Description

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

`[d, sec] = history(c, s, f, FromDate, ToDate, per)` returns the historical data for the field, `f`, for the dates `FromDate` to `ToDate`. `per` specifies the period of the data. For example, `per = {'daily','calendar'}` returns daily data for all calendar days reporting missing data as `NaN`s. `per = {'actual'}` returns the data using the default periodicity and default calendar reporting missing data as `NaN`s. The default periodicity depends on the security. If a security is reported on a monthly basis, the default periodicity is monthly. The default calendar is actual trading days. The possible values of `per` are as follows:

Value	Time Period
`daily`	Daily
`weekly`	Weekly
`monthly`	Monthly
`quarterly`	Quarterly
`semi_annually`	Semi annually
`yearly`	Yearly

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

`[d, sec] = history(c, s, f, FromDate, ToDate, per, cur)` returns the historical data for the security list `s` for the fields `f` for the dates `FromDate` to `ToDate` based on the given currency, `cur`.

`[d, sec] = history(c, s, f, FromDate, ToDate, per, cur, Name, Value, ...)` returns the historical data for the security list `s` for the fields `f` for the dates `FromDate` to `ToDate` based on the given currency, `cur`. `Name,Value` pair arguments are used for additional Bloomberg request settings.

Tips

- Historical requests made before the market opens on the current date that include the current date as the end date may have missing or skewed data. For example, if the `last_price` and `volume` are requested, the `last_price` may not be returned and the `volume` data for the last, current date may be shifted into the `last_price` column.
- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/classpath.txt`. For more information about the static Java class path, see *"The Static Path"* in the MATLAB User's Guide.

Definitions

Anchor Date

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

Examples

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

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

Return the monthly closing price for the given dates for the given security:

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

Return the monthly closing price converted to US dollars for the given dates for the given security:

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

Return the daily closing price converted to US dollars for the given dates for the given security:

```
[d, sec] = history(c, 'ABC US Equity', ...
   'LAST_PRICE', '8/01/2010', '8/10/2010', {'daily',...
   'actual', 'all_calendar_days', 'nil_value'}, 'USD')
```

Return the weekly closing price converted to US dollars for the given dates for the given security. Note that the anchor date is dependent on the date 12/23/1999 in this case. Because this date is a Thursday, each previous value will be reported for the Thursday of the week in question.

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

Return the closing price converted to US dollars for the given dates for the given security using the default period of the data. The default period of a security is dependent on the security itself and not set in this function.

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

Return the closing price converted to US dollars for the given dates for the given security using the default period of the data. The prices are adjusted for normal cash and splits.

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

When specifying Bloomberg override fields, use the `'overrideOption'`. The `overrideOption` 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.

```
reqData3 = history(conn,'AKZA NA Equity', ...
```

```
    'BEST_EPS_MEDIAN', datenum('01.10.2010', ...
    'dd.mm.yyyy'), datenum('30.10.2010','dd.mm.yyyy'), ...
    {'daily','calendar'}, [],'overrideOption', ...
    {'BEST_FPERIOD_OVERRIDE', 'BF'}, 'CapChg', true);
```

See Also

blp | blp.realtime | blp.timeseries | blp.getdata

blp.isconnection

Purpose

`true` if valid Bloomberg V3 connection

Syntax

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

Description

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

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

See Also

`blp` | `blp.close` | `blp.getdata`

Purpose

Bloomberg V3 real-time data retrieval

Syntax

```
d = realtime(c, sec, fields)
[subs, t] = realtime(c, sec, fields, api)
```

Description

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

`[subs, t] = realtime(c, sec, fields, api)` returns the subscription list, `subs`, and the timer, `t`, associated with the real-time callback for the subscription list. Given connection `c`, the `realtime` function subscribes to a security or securities, `sec`, and requests fields, `fields`, to update in real time while running a function, `api`. For information about an example that demonstrates the data import functionality of `realtime`, type `help v3showtrades` at the command line.

Examples

Subscribe to the security `ABC US Equity`. Request that the fields `Last_Trade` and `Volume` update in real time while the function `v3stockticker` is running:

```
realtime(c,'ABC US Equity', ...
   {'Last_Trade','Volume'},'v3stockticker')
```

`realtime` returns only the most recent event—that is, data for a single security—when you use it in snapshot mode with no callback. To get data for multiple securities, use:

```
x = getdata(b, {'IBM US Equity', 'AAPL US EQUITY'}, ...
       {'Last_Trade', 'Volume', 'Open', 'High', 'Low'})
```

See Also

`blp | blp.history | blp.timeseries`

blp.stop

Purpose

Unsubscribe real time requests for Bloomberg V3

Syntax

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

Description

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

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

Purpose

Bloomberg V3 intraday tick data

Syntax

```
d = timeseries(c,s,t)
d = timeseries(c,s,{StartDate,EndDate})
d = timeseries(c,s,t,b,f)
d = timeseries(c,s,t,[],f,{'api'},{'val'})
```

Description

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

`d = timeseries(c,s,{StartDate,EndDate})` returns raw tick data for the date range defined by `StartDate` and `EndDate`.

`d = timeseries(c,s,t,b,f)` returns tick data in intervals of `b` minutes for the field `f`. Intraday tick data requested over a certain interval is returned with columns representing Time, Open, High, Low, Last Price, Volume of Ticks, Number of Ticks, and Total Tick Value in the bar.

`d = timeseries(c,s,t,[],f,{'api'},{'val'})` returns tick data for the field `f`. The cell array of `api` options can include any of `includeConditionCodes`, `includeExchangeCodes`, and `includeBrokerCodes`. You can set the corresponding cell array of values to `true` or `false`.

Tips

- For better performance, add the Bloomberg file `blpapi3.jar` to the MATLAB static Java class path by modifying the file `$MATLAB/toolbox/local/classpath.txt`. For more information about the static Java class path, see *"The Static Path"* in the MATLAB User's Guide.
- You cannot retrieve Bloomberg intraday tick data for a date more than 140 days ago.

Examples

Return today's time series for the given security:

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

The timestamp and tick value are returned.

Return today's Trade tick series for the given security aggregated into 5-minute intervals:

```
d = timeseries(c,'ABC US Equity',floor(now),5,'Trade')
```

Return the Trade tick series for the past 50 days for the given security aggregated into 5-minute intervals:

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

Return the Bid, Ask, and Trade tick series for the security `RIM CT Equity` on June 22, 2011 during a specified 5-minute interval, without specifying the aggregation parameter.

```
d = timeseries(c,'RIM CT Equity',{'06/22/2011 12:15:00',...
   '06/22/2011 12:20:00'},[],{'Bid','Ask','Trade'})
```

Return the Trade tick series for the security `RIM CT Equity` on June 22, 2011 during a specified 5-minute interval. Also return the condition codes, exchange codes, and broker codes.

```
d = timeseries(c,'RIM CT Equity',{'06/22/2011 12:15:00',...
   '06/22/2011 12:20:00'},[],'Trade',...
   {'includeConditionCodes','includeExchangeCodes',...
    'includeBrokerCodes'},{'true','true','true'});
```

See Also

`blp` | `blp.history` | `blp.realtime`

Purpose

Establish connections to Thomson Reuters Datastream API

Syntax

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

Arguments

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

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

Description

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

Examples

Establish a connection to the Thomson Reuters Datastream API:

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

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

See Also

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

datastream.close

Purpose

Close connections to Thomson Reuters Datastream data servers

Syntax

```
close(Connect)
```

Arguments

`Connect`	Thomson Reuters Datastream connection object created with the `datastream` function.

Description

`close(Connect)` closes a connection to a Thomson Reuters Datastream data server.

See Also

`datastream`

Purpose

Request data from Thomson Reuters Datastream data servers

Syntax

```
data = fetch(Connect, 'Security')
data = fetch(Connect, 'Security', 'Fields')
data = fetch(Connect, 'Security', 'Fields', 'Date')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
   'ToDate')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
   'ToDate', 'Period')
data = fetch(Connect, 'Security', 'Fields', 'FromDate',
   'ToDate', 'Period', 'Currency')
```

Arguments

`Connect`	Thomson Reuters Datastream connection object created with the `datastream` function.
`'Security'`	MATLAB string containing the name of a security, or cell array of strings containing names of multiple securities. This data is in a format recognizable by the Thomson Reuters Datastream data server.
`'Fields'`	(Optional) MATLAB string or cell array of strings indicating the data fields for which to retrieve data.
`'Date'`	(Optional) MATLAB string indicating a specific calendar date for which you request data.
`'FromDate'`	(Optional) Start date for historical data.

`'ToDate'`	(Optional) End date for historical data. If you specify a value for `'ToDate'`, `'FromDate'` cannot be an empty value. **Note** You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.
`'Period'`	(Optional) Period within a date range. `Period` values are: • `'d'`: daily values • `'w'`: weekly values • `'m'`: monthly values
`'Currency'`	(Optional) Currency in which `fetch` returns the data.

Note You can enter the optional arguments `'Fields'`, `'FromDate'`, `'ToDate'`, `'Period'`, and `'Currency'` as MATLAB strings or empty arrays (`[ ]`).

Description

`data = fetch(Connect, 'Security')` returns the default time series for the indicated security.

`data = fetch(Connect, 'Security', 'Fields')` returns data for the specified security and fields.

`data = fetch(Connect, 'Security', 'Fields', 'Date')` returns data for the specified security and fields on a particular date.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')` returns data for the specified security and fields for the indicated date range.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period')` returns instrument data for the given range with the indicated period.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate', 'Period', 'Currency')` also specifies the currency in which to report the data.

Note The Thomson Reuters Datastream interface returns all data as strings. For example, it returns `Price` data to the MATLAB workspace as a cell array of strings within the structure. There is no way to determine the data type from the Datastream interface.

Examples

Retrieving Time Series Data

Return the trailing one-year price time series for the instrument `ICI`, with the default value `P` for the `'Fields'` argument using the command:

```
data = fetch(Connect, 'ICI')
```

Or the command:

```
data = fetch(Connect, 'ICI', 'P')
```

Retrieving Opening and Closing Prices

Return the closing and opening prices for the instruments `ICI` on the date September 1, 2007.

```
data = fetch(Connect, 'ICI', {'P', 'PO'}, '09/01/2007')
```

Retrieving Monthly Opening and Closing Prices for a Specified Date Range

Return the monthly closing and opening prices for the securities `ICI` and `IBM` from 09/01/2005 to 09/01/2007:

```
data = fetch(Connect, {'ICI', 'IBM'}, {'P', 'PO'}, ...
'09/01/2005', '09/01/2007', 'M')
```

Retrieving Static Data

Return the static fields `NAME` and `ISIN`:

```
data = fetch(Connect,{'IBM~REP'}, {'NAME','ISIN'});
```

You can also return `SECD` in this way.

See Also

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

Purpose

Retrieve properties of Thomson Reuters Datastream connection objects

Syntax

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

Arguments

`Connect`	Thomson Reuters Datastream connection object created with the `datastream` function.
`PropertyName`	(Optional) A MATLAB string or cell array of strings containing property names. Valid property names include: • `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

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

datastream.isconnection

Purpose

Verify whether connections to Thomson Reuters Datastream data servers are valid

Syntax

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

Arguments

`Connect`	Thomson Reuters Datastream connection object created with the `datastream` function.

Description

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

Examples

Establish a connection to the Thomson Reuters Datastream API:

```
c = datastream
```

Verify that `c` is a valid connection:

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

See Also

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

Purpose

eSignal Desktop API connection

Syntax

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

Description

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

Examples

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

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

Note that your path may differ from the one shown. Look for `Interop.IESignal.dll` in the installation directory of eSignal. You must have access to eSignal to use this functionality.

Create an eSignal connection handle:

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

See Also

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

esig.close

Purpose Close eSignal connection

Syntax `close(e)`

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

Purpose

Current eSignal data

Syntax

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

Description

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

Examples

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

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

See Also

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

esig.getfundamentaldata

Purpose

Current eSignal fundamenal data

Syntax

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

Description

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

Examples

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

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

See Also

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

Purpose

eSignal historical data

Syntax

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

Description

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

Examples

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

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

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

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

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

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

See Also

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

esig.timeseries

Purpose

eSignal intraday tick data

Syntax

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

Description

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

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

Tips

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

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

Examples

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

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

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

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

Return five days of raw intraday tick data. All fields are returned in the following order: `TickType`, `Time`, `Price`, `Size`, `Exchange`, and `Flags`.

```
D = timeseries(E,'ABC US Equity',[],floor(now)-5)
```

See Also

`esig` | `esig.close` | `esig.getdata` | `esig.history`

factset

Purpose

Establish connections to FactSet data servers

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 FAST interface.

Examples

Establish a connection to a FactSet server:

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

See Also

`factset.close` | `factset.fetch` | `factset.get` | `factset.isconnection`

Purpose

Close connections to FactSet data servers

Syntax

```
close(Connect)
```

Arguments

`Connect`	FactSet connection object created with the `factset` function.

Description

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

See Also

`factset`

factset.fetch

Purpose

Request data from FactSet data servers

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`Library`	FactSet formula library.
`Security`	A MATLAB string or cell array of strings containing the names of securities in a format recognizable by the FactSet server.
`Fields`	A MATLAB string or cell array of strings indicating the data fields for which to retrieve data.
`Date`	Date string or serial date number indicating date for the requested data. If you enter today's date, `fetch` returns yesterday's data.
`FromDate`	Beginning date for date range.

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

`ToDate`	End date for date range.
`Period`	Period within date range. `Period` values are: • `'d'`: daily values • `'b'`: business day daily values • `'m'`: monthly values • `'mb'`: beginning monthly values • `'me'`: ending monthly values • `'q'`: quarterly values • `'qb'`: beginning quarterly values • `'qe'`: ending quarterly values • `'y'`: annual values • `'yb'`: beginning annual values • `'ye'`: ending annual values

Description

`data = fetch(Connect)` returns the names of all available formula libraries.

`data = fetch(Connect, 'Library')` returns the valid field names for a given formula library.

`data = fetch(Connect, 'Security', 'Fields')` returns data for the specified security and fields.

`data = fetch(Connect, 'Security', 'Fields', 'Date')` returns security data for the specified fields on the requested date.

`data = fetch(Connect, 'Security', 'Fields', 'FromDate', 'ToDate')` returns security data for the specified fields for the date range `FromDate` to `ToDate`.

`data = fetch(Connect, 'Security', 'FromDate', 'ToDate', 'Period')` returns security data for the date range `FromDate` to `ToDate` with the specified period.

Examples

Retrieving Names of Available Formula Libraries

Obtain the names of available formula libraries:

```
D = fetch(Connect)
```

Retrieving Valid Field Names of a Specified Library

Obtain valid field names of the `FactSetSecurityCalcs` library:

```
D = fetch(Connect, 'fs')
```

Retrieving the Closing Price of a Specified Security

Obtain the closing price of the security `IBM`:

```
D = fetch(Connect, 'IBM', 'price')
```

Retrieving the Closing Price of a Specified Security Using Default Date Period

Obtain the closing price for `IBM` using the default period of the data:

```
D = fetch(C, 'IBM', 'price', '09/01/07', '09/10/07')
```

Retrieving the Monthly Closing Prices of a Specified Security for a Given Date Range

Obtain the monthly closing prices for `IBM` from 09/01/05 to 09/10/07:

```
D = fetch(C, 'IBM', 'price', '09/01/05', '09/10/07', 'm')
```

See Also

`factset.close` | `factset` | `factset.isconnection`

Purpose

Retrieve properties of FactSet connection objects

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.
`PropertyName`	(Optional) A MATLAB string or cell array of strings containing property names. Property names are: • `user` • `serial` • `password` • `cid`

Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the FactSet connection object.

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

Examples

Establish a connection to a FactSet data server:

```
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

`factset.close` | `factset.fetch` | `factset` | `factset.isconnection`

Purpose

Verify whether connections to FactSet data servers are valid

Syntax

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

Arguments

`Connect`	FactSet connection object created with the `factset` function.

Description

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

Examples

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

```
c = factset
```

Verify that `c` is a valid connection:

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

See Also

`factset.close` | `factset.fetch` | `factset` | `factset.get`

fred

Purpose

Connect to FRED data servers

Syntax

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

Arguments

`URL`	Create a connection using a specified URL.

Description

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

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

Examples

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

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

See Also

`fred.close` | `fred.fetch` | `fred.get` | `fred.isconnection`

Purpose

Close connections to FRED data servers

Syntax

```
close(Connect)
```

Arguments

`Connect`	FRED connection object created with the `fred` function.

Description

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

Examples

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

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

Close this connection:

```
close(c)
```

See Also

`fred`

fred.fetch

Purpose

Request data from FRED data servers

Syntax

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

Arguments

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

Description

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

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

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

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

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

Examples

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

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

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

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

See Also

`fred.close` | `fred.get` | `fred.isconnection`

fred.get

Purpose

Retrieve properties of FRED connection objects

Syntax

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

Arguments

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

Description

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

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

Examples

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

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

Retrieve the port and IP address for the connection:

```
p = get(c, {'port', 'ip'})
p =
     port: 8194
     ip: 111.222.33.444
```

See Also

`fred.close` | `fred.fetch` | `fred.isconnection`

Purpose

Verify whether connections to FRED data servers are valid

Syntax

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

Arguments

`Connect`	FRED connection object created with the `fred` function.

Description

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

Examples

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

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

Verify that `c` is a valid connection:

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

See Also

`fred.close` | `fred.fetch` | `fred.get`

haver

Purpose

Connect to local Haver Analytics database

Syntax

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

Arguments

`Databasename`	Local path to the Haver Analytics database.

Description

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

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

`haver.close` | `haver.fetch` | `haver.get` | `haver.isconnection`

Purpose

Set Haver Analytics aggregation mode

Syntax

```
X = aggregation (C)
X = aggregation (C,V)
```

Description

`X = aggregation (C)` returns the current aggregation mode.

`X = aggregation (C,V)` sets the current aggregation mode to `V`. The following table lists possible values for `V`.

Value of `V`	Aggregation mode	Behavior of `aggregation` function
`0`	`strict`	`aggregation` does not fill in values for missing data.
`1`	`relaxed`	`aggregation` fills in missing data based on data available in the requested period.
`2`	`forced`	`aggregation` fills in missing data based on some past value.
`-1`	Not recognized	`aggregation` resets `V` to its last valid setting.

See Also

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

haver.close

Purpose

Close Haver Analytics database

Syntax

```
close(H)
```

Arguments

`H`	Haver Analytics connection object created with the `haver` function.

Description

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

Examples

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

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

Close the connection:

```
close(H)
```

See Also

`haver`

Purpose

Request data from Haver Analytics database

Syntax

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

Arguments

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

Description

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

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

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

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

Examples

Establish a Connection to a Haver Analytics Database

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

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

Retrieving Variable Data

Return data for the variable `FFED`:

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

Retrieving Variable Data for a Specified Date Range

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

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

Retrieving Monthly Variable Data for a Specified Date Range

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

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

See Also

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

Purpose

Retrieve properties from Haver Analytics connection objects

Syntax

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

Arguments

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

Description

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

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

Examples

Establish a Haver Analytics connection, `HDAILY`:

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

Retrieve the name of the Haver Analytics database:

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

See Also

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

haver.info

Purpose

Retrieve information about Haver Analytics variables

Syntax

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

Arguments

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

Description

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

Examples

Establish a Haver Analytics connection `H`:

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

Request information for the variable after `FFED`:

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

The following output is returned:

```
     VarName: 'FFED2'
   StartDate: '01-Jan-1991'
     EndDate: '31-Dec-1998'
   NumberObs: 2088
   Frequency: 'D'
 DateTimeMod: '02-Apr-2007 20:46:37'
   Magnitude: 0
DecPrecision: 2
     DifType: 1
     AggType: 'AVG'
    DataType: '%'
       Group: 'Z05'
      Source: 'FRB'
```

```
  Descriptor: 'Federal Funds [Effective] Rate (% p.a.)'
 ShortSource: 'History'
  LongSource: 'Historical Series'
```

See Also

haver.close | haver.get | haver.isconnection | haver | haver.nextinfo

haver.isconnection

Purpose

Verify whether connections to Haver Analytics data servers are valid

Syntax

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

Arguments

`H`	Haver Analytics connection object created with the `haver` function.

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

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

Purpose

Retrieve information about next Haver Analytics variable

Syntax

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

Arguments

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

Description

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

Examples

Establish a Haver Analytics connection `H`:

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

Request information for the variable following `FFED`:

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

The following structure is returned:

```
     VarName: 'FFED2'
   StartDate: '01-Jan-1991'
     EndDate: '31-Dec-1998'
   NumberObs: 2088
   Frequency: 'D'
 DateTimeMod: '02-Apr-2007 20:46:37'
   Magnitude: 0
DecPrecision: 2
     DifType: 1
     AggType: 'AVG'
    DataType: '%'
       Group: 'Z05'
      Source: 'FRB'
```

```
  Descriptor: 'Federal Funds [Effective] Rate (% p.a.)'
 ShortSource: 'History'
  LongSource: 'Historical Series'
```

See Also

haver.close | haver.get | haver | haver.info | haver.isconnection

Purpose

Run Haver Analytics graphical user interface (GUI)

Syntax

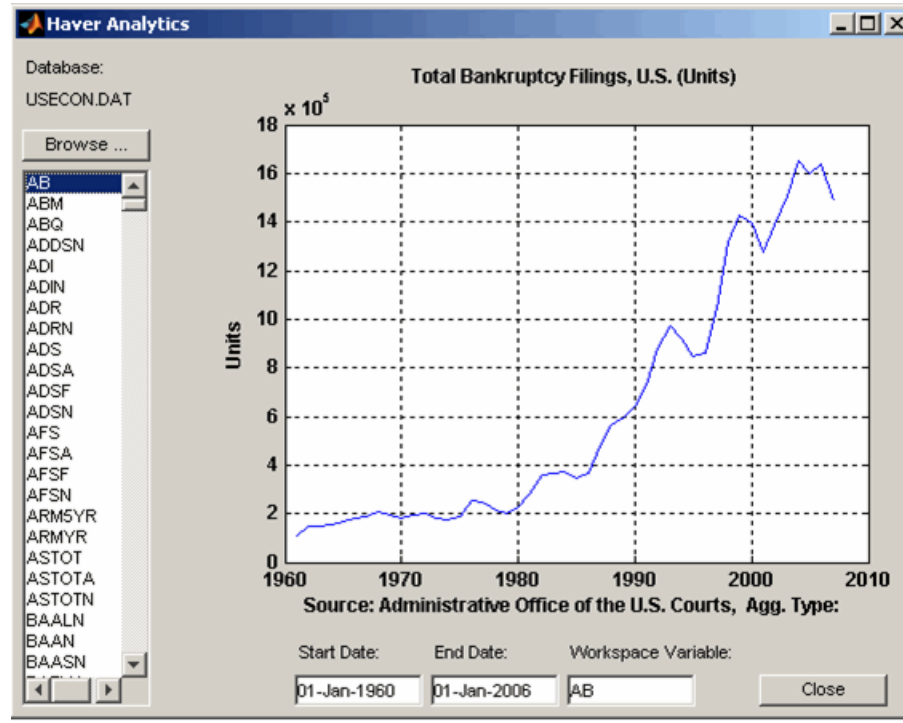
```
havertool(H)
```

Arguments

`H`	Haver Analytics connection object created with the `haver` function.

Description

`havertool(H)` runs the Haver Analytics graphical user interface (GUI). The GUI appears in the following figure.



The GUI fields and buttons are:

- **Database**: The currently selected Haver Analytics database.
- **Browse**: Allows you to browse for Haver Analytics databases, and populates the variable list with the variables in the database you specify.
- **Start Date**: The data start date of the selected variable.
- **End Date**: The data end date of the selected variable.
- **Workspace Variable**: The MATLAB variable to which `havertool` writes data for the currently selected Haver Analytics variable.
- **Close**: Closes all current connections and the Haver Analytics GUI.

Examples

Establish a Haver Analytics connection `H`:

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

Open the graphical user interface (GUI) demonstration:

```
havertool(H)
```

See Also

`haver`

Purpose

Connect to Interactive Data Pricing and Reference Data's RemotePlus data servers

Syntax

```
Connect = idc
```

Description

`Connect = idc` connects to the Interactive Data Pricing and Reference Data's RemotePlus server. `Connect` is a connection handle used by other functions to obtain data.

Examples

Connect to an Interactive Data Pricing and Reference Data's RemotePlus server:

```
c = idc
```

See Also

`idc.close` | `idc.fetch` | `idc.get` | `idc.isconnection`

idc.close

Purpose

Close connections to Interactive Data Pricing and Reference Data's RemotePlus data servers

Syntax

```
close(Connect)
```

Arguments

`Connect`	Interactive Data Pricing and Reference Data's RemotePlus connection object created with the `idc` function.

Description

`close(Connect)` closes the connection to the Interactive Data Pricing and Reference Data's RemotePlus server.

Examples

Establish an Interactive Data Pricing and Reference Data's RemotePlus connection, `c`:

```
c = idc
```

Close this connection:

```
close(c)
```

See Also

`idc`

Purpose

Request data from Interactive Data Pricing and Reference Data's RemotePlus data servers

Syntax

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

Arguments

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

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

`'ToDate'`	End date for historical data.

`'Period'`	Period within date range.
`'GUICategory'`	GUI category. Possible values are: • `'F'` (All valid field categories) • `'S'` (All valid security categories)

Description

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

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

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

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

Examples

Open the dialog box to look up securities:

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

Open the dialog box to select fields:

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

See Also

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

Purpose

Retrieve properties of Interactive Data Pricing and Reference Data's RemotePlus connection objects

Syntax

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

Arguments

`Connect`	Interactive Data Pricing and Reference Data's RemotePlus connection object created with the `idc` function.
`PropertyName`	(Optional) A MATLAB string or cell array of strings containing property names. Property names are: • `'Connected'` • `'Connection'` • `'Queued'`

Description

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

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

See Also

`idc.close` | `idc` | `idc.isconnection`

idc.isconnection

Purpose

Verify whether connections to Interactive Data Pricing and Reference Data's RemotePlus data servers are valid

Syntax

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

Arguments

`Connect`	Interactive Data Pricing and Reference Data's RemotePlus connection object created with the `idc` function.

Description

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

Examples

Establish an Interactive Data Pricing and Reference Data's RemotePlus connection `c`:

```
c = idc
```

Verify that `c` is a valid connection:

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

See Also

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

Purpose

Connect to Kx Systems, Inc. kdb+ databases

Syntax

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

Arguments

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

Description

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

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

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

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

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

Examples

Run the following command from a DOS prompt to specify the port number `5001`:

```
q tradedata.q -p 5001
```

Connect to a Kx Systems, Inc. server using IP address `LOCALHOST` and port number `5001`:

```
k = kx('LOCALHOST',5001)
handle: [1x1 c]
        ipaddress: 'localhost'
        port: 5001
```

See Also

`kx.close` | `kx.exec` | `kx.get` | `kx.fetch` | `kx.tables`

Purpose

Close connections to Kx Systems, Inc. kdb+ databases

Syntax

```
close(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with the `kx` function.

Description

`close(k)` closes the connection to the Kx Systems, Inc. kdb+ database.

Examples

Close the connection, `k`, to the Kx Systems, Inc. kdb+ database:

```
close(k)
```

See Also

`kx`

kx.exec

Purpose

Run Kx Systems, Inc. kdb+ commands

Syntax

```
exec(k,command)
exec(k,command,p1,p2,p3)
exec(k,command,p1)
exec(k,command,p1,p2)
exec(k,command,p1,p2,p3)
exec(k,command,p1,p2,p3,sync)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with the `kx` function.
`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

`kx.fetch` | `kx.insert` | `kx`

kx.fetch

Purpose

Request data from Kx Systems, Inc. kdb+ databases

Syntax

```
d = fetch(k,ksql)
d = fetch(k,ksql,p1,p2,p3)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with the `kx` function.
`ksql`	The Kx Systems, Inc. kdb+ command.
`p1,p2,p3`	Input parameters for the `ksql` command.

Description

`d = fetch(k,ksql)` returns data from a Kx Systems, Inc. kdb+ database in a MATLAB structure where `k` is the Kx Systems, Inc. kdb+ object and `ksql` is the Kx kdb+ command. `ksql` can be any valid kdb+ command. The output of this method 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

`kx.exec` | `kx.insert` | `kx`

kx.get

Purpose

Retrieve Kx Systems, Inc. kdb+ connection object properties

Syntax

```
v = get(k,'PropertyName')
v = get(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with the `kx` function.
`'PropertyName'`	A string or cell array of strings containing property names. The property names are: • `'handle'` • `'ipaddress'` • `'port'`

Description

`v = get(k,'PropertyName')` returns a MATLAB structure containing the value of the specified properties for the Kx Systems, Inc. kdb+ connection object.

`v = get(k)` returns a MATLAB structure where each field name is the name of a property of `k` and the associated value of the property.

Examples

Get the properties of the connection to the Kx Systems, Inc. kdb+ database, `K`:

```
v = get(k)
v =
   handle:  [1x1 c]
       ipaddress: 'localhost'
       port: '5001'
```

See Also

`kx.close` | `kx.exec` | `kx.fetch` | `kx.insert` | `kx`

Purpose

Write data to Kx Systems, Inc. kdb+ databases

Syntax

```
insert(k,tablename,data)
x = insert(k,tablename,data,sync)
```

Arguments

`k`	The Kx Systems, Inc. kdb+ connection object created with the `kx` function.
`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

`kx.close` | `kx.fetch` | `kx.get` | `kx.tables`

kx.isconnection

Purpose

Verify whether connections to Kx Systems, Inc. kdb+ databases are valid

Syntax

```
x = isconnection(k)
```

Arguments

`k`	Kx Systems, Inc. kdb+ connection object created with the `kx` function.

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

`kx.close` | `kx.fetch` | `kx.get` | `kx`

Purpose

Retrieve table names from Kx Systems, Inc. kdb+ databases

Syntax

```
t = tables(k)
```

Arguments

`k`	The Kx Systems, Inc. kdb+ connection object created with the `kx` function.

Description

`t = tables(k)` returns the list of tables for the Kx Systems, Inc. kdb+ connection.

Examples

Retrieve table information for the Kx Systems, Inc. kdb+ database using the connection `k`:

```
t = tables(k)
t =
     'intraday'
   'seclist'
     'trade'
```

See Also

`kx.exec` | `kx.fetch` | `kx.insert` | `kx`

Purpose

Connect to Thomson Reuters Tick History

Syntax

```
r = rdth(username,password)
```

Description

`r = rdth(username,password)` creates a Thomson Reuters Tick History connection to enable intraday tick data retrieval.

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 intra day 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)
```

See Also

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

Purpose

Close Thomson Reuters Tick History connection

Syntax

```
close(r)
```

Description

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

See Also

`rdth`

rdth.fetch

Purpose

Request Thomson Reuters Tick History data

Syntax

```
x = fetch(r,sec)
x = fetch(r,sec,tradefields,daterange,reqtype,messtype,
   exchange,domain)
x = fetch(r,sec,tradefields,daterange,reqtype,messtype,
   exchange,domain,marketdepth)
```

Description

`x = fetch(r,sec)` returns information about the security, `sec`, such as the code, currency, exchange, and name. `r` is the Thomson Reuters Tick History connection object.

`x = fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain)` returns data for the request security, `sec`, based on the type request and message types, `reqtype` and `messtype`, respectively. Data for the fields specified by `tradefields` is returned for the data range bounded by `daterange`. Specifying the `exchange` of the given security improves the speed of the data request. `domain` specifies the security type.

`x = fetch(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain,market` additionally specifies the depth of level 2 data, `marketdepth`, to return for a `'MarketDepth'` request type. `marketdepth` must be a numeric value between 1 and 10, returning up to 10 bid/ask values for a given security.

Note Do not use date ranges for end of day requests. You can specify a range of hours on a single day, but not a multiple day range.

Tips

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

Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```

returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get information pertaining to a particular security, the command

```
d = fetch(r,'GOOG.O',{'Volume','Price','Exchange ID'},...
{'09/05/2008 12:00:00','09/05/2008 12:01:00'},...
'TimeAndSales','Trade','NSQ','EQU')
```

returns data starting with (not all data is shown):

```
d =
'#RIC'      'Date[L]'        'Time[L]'         'Type'...
     'Ex/Cntrb.ID'    'Price'
'GOOG.O'    '05-SEP-2008'    '12:00:01.178'    'Trade'...
    'NAS'            '443.86'
'Volume'
'200'
```

The command

```
d = fetch(r,'GOOG.O',{'Volume','Last'},{'09/05/2008'},...
'EndOfDay','End Of Day','NSQ','EQU')
```

returns

```
d =
    '#RIC'      'Date[L]'        'Time[L]'       ...
  'Type'    'Last'     'Volume'
    'GOOG.O'    '05-SEP-2008'    '23:59:00.000'   ...
```

```
 'End Of Day'    '444.25'    '4538375'
```

For

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

for example, the exchange of the security is `x.Exchange` or `NSQ`. To determine the asset domain of the security, use the value of `x.Type`, in this case 113. Using the information from `v = get(r)`,

```
j = find(v.InstrumentTypes.Value == 113)
```

returns

```
j =46
```

The command

```
v.InstrumentTypes.Value(j)
```

returns

```
ans =
   113
```

The command

```
v.InstrumentTypes.Name(j)
```

returns

```
ans =
    'Equities'
```

The command

```
v.AssetDomains.Value(strcmp(v.InstrumentTypes.Name(j),...
v.AssetDomains.Name))
```

returns

```
ans =
    'EQU'
```

Knowing the security exchange and domain helps the interface to resolve the security symbol and return data more quickly.

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

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

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = fetch(R,'AAPL.O',{'Bid Price','Bid Size'},...
         {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

See Also

`rdth | rdth.close | rdth.get`

rdth.get

Purpose

Get Thomson Reuters Tick History connection properties

Syntax

```
v = get(r,'propertyname')
v = get(r)
```

Description

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

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

Properties include:

- `AssetDomains`
- `BondTypes`
- `Class`
- `Countries`
- `CreditRatings`
- `Currencies`
- `Exchanges`
- `FuturesDeliveryMonths`
- `InflightStatus`
- `InstrumentTypes`
- `MessageTypes`
- `OptionExpiryMonths`
- `Quota`
- `RestrictedPEs`
- `Version`

Examples

To create a Thomson Reuters Tick History connection, the command

```
 r = rdth('user@company.com','mypassword')
```

returns

```
r =
client: [1x1 com.thomsonreuters.tickhistory. ...
webservice.client.RDTHApiClient]
user: 'user@company.com'
password: '**********'
```

To get a listing of properties for the `rdth` connection, the command

```
v = get(r)
```

returns

```
v =

             AssetDomains: [1x1 struct]
                BondTypes: {255x1 cell}
                    Class: 'class com.thomsonreuters. ...
tickhistory.webservice.client.RDTHApiClient'
                Countries: {142x1 cell}
            CreditRatings: {82x1 cell}
               Currencies: [1x1 struct]
                Exchanges: [1x1 struct]
    FuturesDeliveryMonths: {12x1 cell}
           InflightStatus: [1x1 com.thomsonreuters. ...
tickhistory.webservice.types.InflightStatus]
          InstrumentTypes: [1x1 struct]
             MessageTypes: [1x1 struct]
       OptionExpiryMonths: {12x1 cell}
                    Quota: [1x1 com.thomsonreuters. ...
tickhistory.webservice.types.Quota]
            RestrictedPEs: {2758x1 cell}
                  Version: [1x1 com.thomsonreuters. ...
```

```
tickhistory.webservice.types.Version]
```

See Also

rdth | rdth.fetch

Purpose

Verify whether Thomson Reuters Tick History connections are valid

Syntax

```
x = isconnection(r)
```

Description

`x = isconnection(r)` returns 1 if `r` is a valid `rdth` client and 0 otherwise.

Examples

Verify that `r` is a valid connection:

```
r = rdth('user@company.com','mypassword');
x = isconnection(r)
x = 1
```

See Also

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

rdth.status

Purpose

Status of FTP request for Thomson Reuters Tick History data

Syntax

```
[s,qp] = status(r,x)
```

Description

`[s,qp] = status(r,x)` returns the status and queue position of the Thomson Reuters Tick History (TRTH) FTP request handle, `x`. When `s` is equal to `'Complete'`, download the file from the TRTH server manually or programmatically.

Examples

Check the status of your FTP request:

```
x = submitftp(r,'GOOG.O',{'Exchange ID','Price','Volume'}, ...
   {(floor(now))-10,(floor(now))},'TimeAndSales','Trade', ...
   'NSQ','EQU')

s = [];
while ~strcmp(s,'Complete')
[s,qp] = status(r,x);
end
```

Optionally, download the file from the TRTH server programmatically. The data file is generated in a directory, `api-results`, on the server. The file has extension `csv.gz`.

```
filename = ['/api-results/' char(x) '-report.csv.gz'];
urlwrite(['https://tickhistory.thomsonreuters.com/HttpPull/Download?'...
          'user=' username '&pass=' password '&file=' filename''],...
          'rdth_results.csv.gz');
```

This call to `urlwrite` saves the downloaded file with the name `rdth_results.csv.gz` in the current directory.

Purpose

Submit FTP request for Thomson Reuters Tick History data

Syntax

```
x = submitftp(r, sec)
x = submitftp(r, sec, tradefields, daterange, reqtype,
   messtype, exchange, domain)
x = submitftp(r,sec,tradefields,daterange,reqtype,messtype,
   exchange,domain,marketdepth)
```

Description

`x = submitftp(r, sec)` returns information about the security, `sec`, such as the code, currency, exchange, and name for the given `trth` connection object, `r`.

`x = submitftp(r, sec, tradefields, daterange, reqtype, messtype, exchange, domain)` submits an FTP request for the request security, `sec`, based on the type request and message types, `reqtype` and `messtype`, respectively. Data for the fields specified by `tradefields` is returned for the data range bounded by `daterange`. Specifying the `exchange` or the given security improves the speed of the data request. `domain` specifies the security type.

`x = submitftp(r,sec,tradefields,daterange,reqtype,messtype,exchange,domain,marketdepth)` additionally specifies the depth of level 2 data, `marketdepth`, to return for a `'MarketDepth'` request type. `marketdepth` must be a numeric value between 1 and 10, returning up to 10 bid/ask values for a given security.

To monitor the status of the FTP request, enter the command

```
[s,qp] = status(r,x)
```

The `status` function returns a status message and queue position. When `S = 'Complete'`, download the resulting compressed `.csv` file from the `TRTH` servers. Once the `.csv` file has been saved to disk, use `rdthloader('filename')` to load the data into the MATLAB workspace. To obtain more information request and message types and their associated field lists, use the command `get(r)`.

Examples

Specify parameters for FTP request:

```
submitftp(r,{'IBM.N','GOOG.O'}, ...
   {'Open','Last','Low','High'}, ...
   {floor(now)-100,floor(now)}, ...
   'EndOfDay','End Of Day','NSQ','EQU');
```

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

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

To use a `'MarketDepth'` level of 3, enter:

```
AaplTickData = submitftp(R,'AAPL.O',{'Bid Price','Bid Size'},...
         {now-.05,now},'MarketDepth','Market Depth','NSQ','EQU',3);
```

See Also

`rdth.fetch` | `rdth.get` | `rdth` | `rdthloader` | `rdth.status`

Purpose

Retrieve data from Thomson Reuters Tick History file

Syntax

```
x = rdthloader(file)
x = rdthloader(file,'date',{DATE1})
x = rdthloader(file,'date',{DATE1, DATE2})
x = rdthloader(file,'security',{SECNAME})
x = rdthloader(file,'start',STARTREC)
x = rdthloader(file,'records', NUMRECORDS)
```

Arguments

Specify the following arguments as name-value pairs. You can specify any combination of name-value pairs in a single call to `rdthloader`.

`file`	Thomson Reuters Tick History file from which to retrieve data.
`'date'`	Use this argument with `{DATE1, DATE2}` to retrieve data between and including the specified dates. Specify the dates as numbers or strings.
`'security'`	Use this argument to retrieve data for `SECNAME`, where `SECNAME` is a cell array containing a list of security identifiers for which to retrieve data.
`'start'`	Use this argument to retrieve data beginning with the record `STARTREC`, where `STARTREC` is the record at which `rdthloader` begins to retrieve data. Specify `STARTREC` as a number.
`'records'`	Use this argument to retrieve `NUMRECORDS` number of records.

Description

`x = rdthloader(file)` retrieves tick data from the Thomson Reuters Tick History file `file` and stores it in the structure `x`.

`x = rdthloader(file,'date',{DATE1})` retrieves tick data from `file` with date stamps of value `DATE1`.

`x = rdthloader(file,'date',{DATE1, DATE2})` retrieves tick data from `file` with date stamps between `DATE1` and `DATE2`.

`x = rdthloader(file,'security',{SECNAME})` retrieves tick data from `file` for the securities specified by `SECNAME`.

`x = rdthloader(file,'start',STARTREC)` retrieves tick data from `file` beginning with the record specified by `STARTREC`.

`x = rdthloader(file,'records', NUMRECORDS)` retrieves `NUMRECORDS` number of records from `file`.

Examples

Retrieve all ticks from the file `file.csv` with date stamps of `02/02/2007`:

```
x = rdthloader('file.csv','date',{'02/02/2007'})
```

Retrieve all ticks from `file.csv` between and including the dates `02/02/2007` and `02/03/2007`:

```
 x = rdthloader('file.csv','date',{'02/02/2007',...
'02/03/2007'})
```

Retrieve all ticks from `file.csv` for the security `XYZ.O`:

```
 x = rdthloader('file.csv','security',{'XYZ.O'})
```

Retrieve the first 10,000 tick records from `file.csv`:

```
 x = rdthloader('file.csv','records',10000)
```

Retrieve data from `file.csv`, starting at record 100,000:

```
  x = rdthloader('file.csv','start',100000)
```

Retrieve up to 100,000 tick records from `file.csv`, for the securities `ABC.N` and `XYZ.O`, with date stamps between and including the dates `02/02/2007` and `02/03/2007`:

```
x = rdthloader('file.csv','records',100000,...
               'date',{'02/02/2007','02/03/2007'},...
```

```
                                'security',{'ABC.N','XYZ.O'})
```

See Also

reuters | rnseloader

Purpose

Create Reuters sessions

Syntax

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

Arguments

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

Description

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

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

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

Examples

Connecting to Reuters Data Servers

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

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

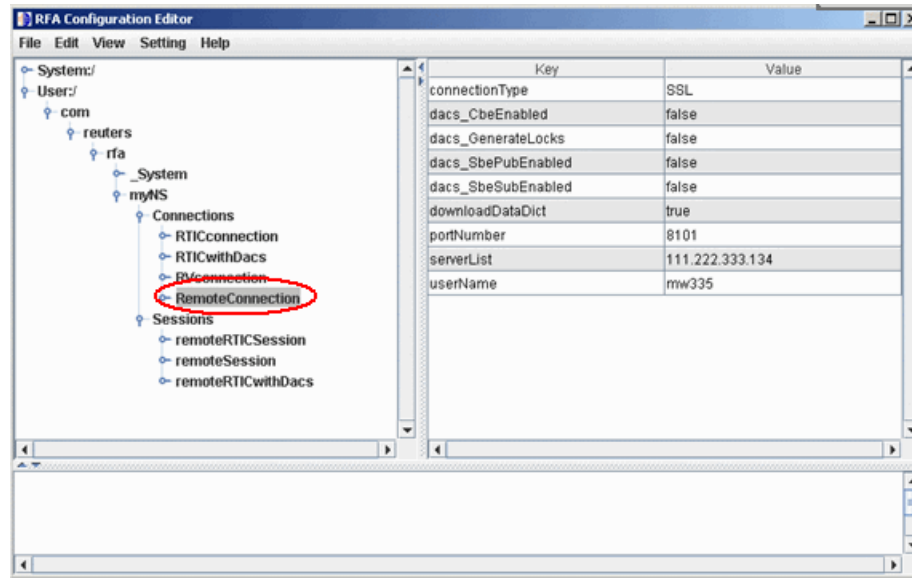
Note If you do not use the Reuters DACS authentication functionality, the following error message appears:

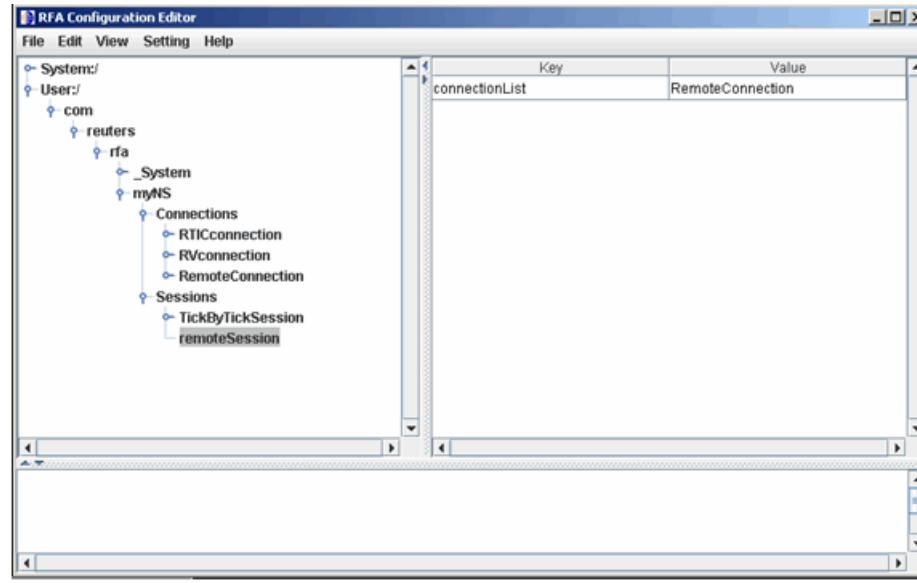
```
com.reuters.rfa.internal.connection.ConnectionImpl
initializeEntitlementsINFO:
com.reuters.rfa.connection.ssl.myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

Connecting to Reuters Data Servers Using DACS Authentication

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

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





2 Add the following to your connection configuration:

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

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

```
dacs_GenerateLocks=false
```

Connecting to Reuters Data Servers Without DACS Authentication

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

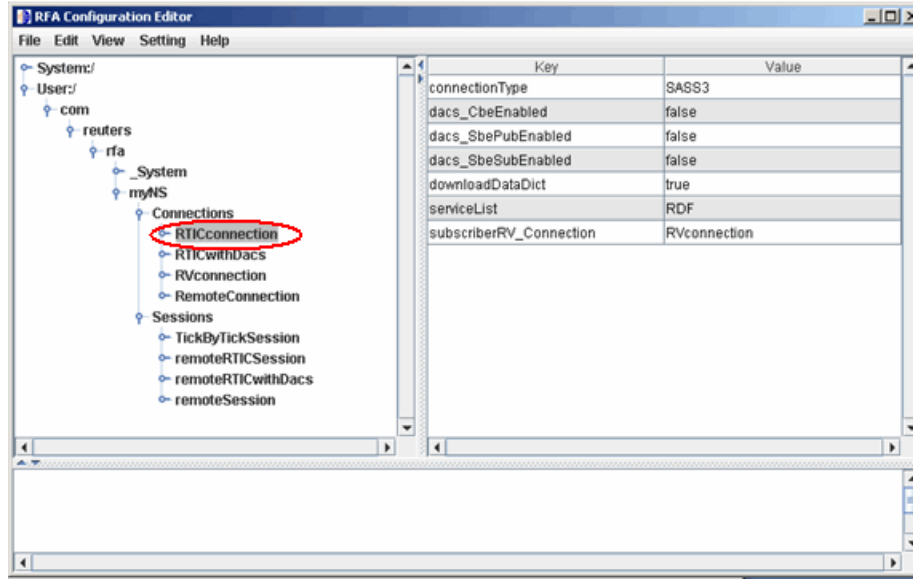
```
r = reuters ('myNS::remoteSession', 'dIDN_RDF')
```

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

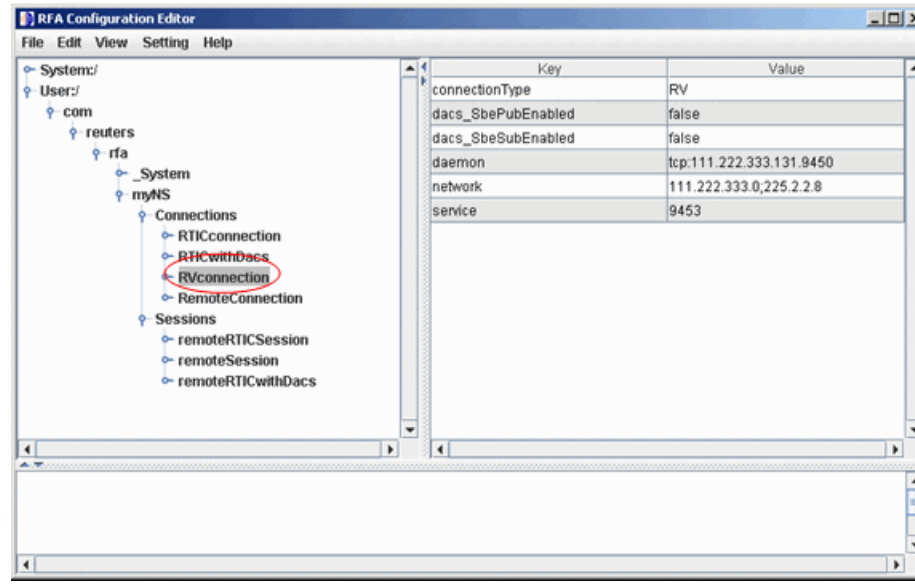
- **Non-DACs-enabled**

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

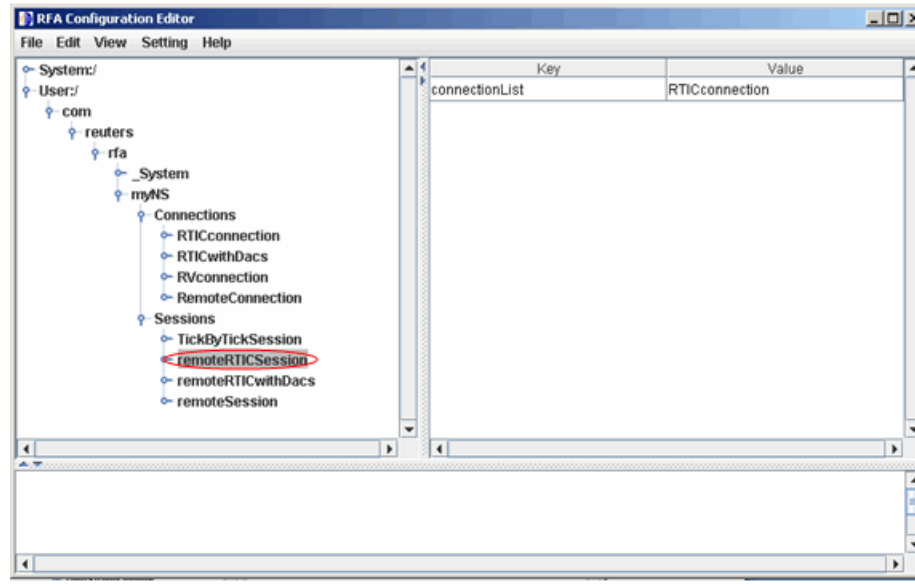
  ```
  r = reuters ('myNS::remoteRTICSession','IDN_RDF')
  ```



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



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


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

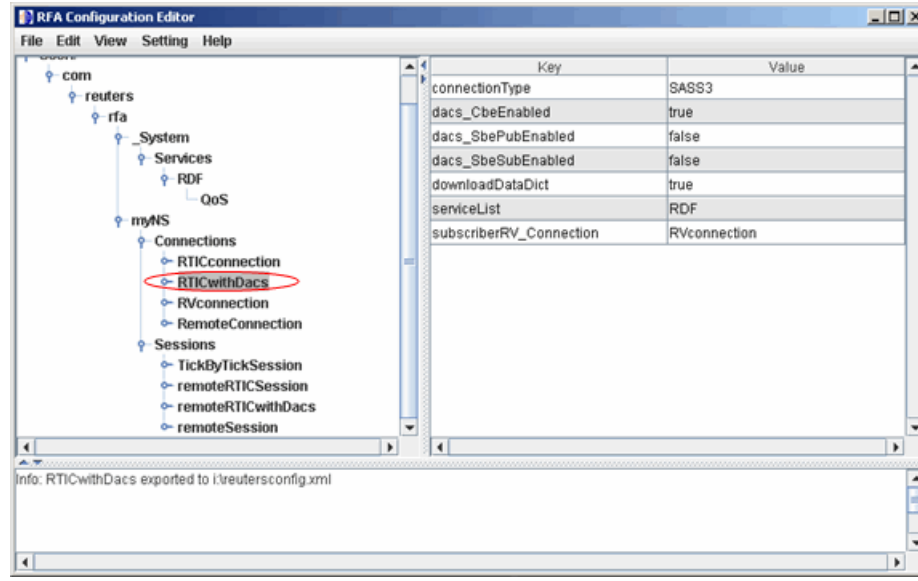
Messages like the following may appear in the MATLAB Command Window when you establish a non-DACs-enabled connection. These messages are informational and can safely be ignored.

```
Oct 5, 2007 2:28:31 PM
com.reuters.rfa.internal.connection.
ConnectionImpl initializeEntitlements
INFO: com.reuters.rfa.connection.ssl....
   myNS.RemoteConnection
DACS disabled for connection myNS::RemoteConnection
```

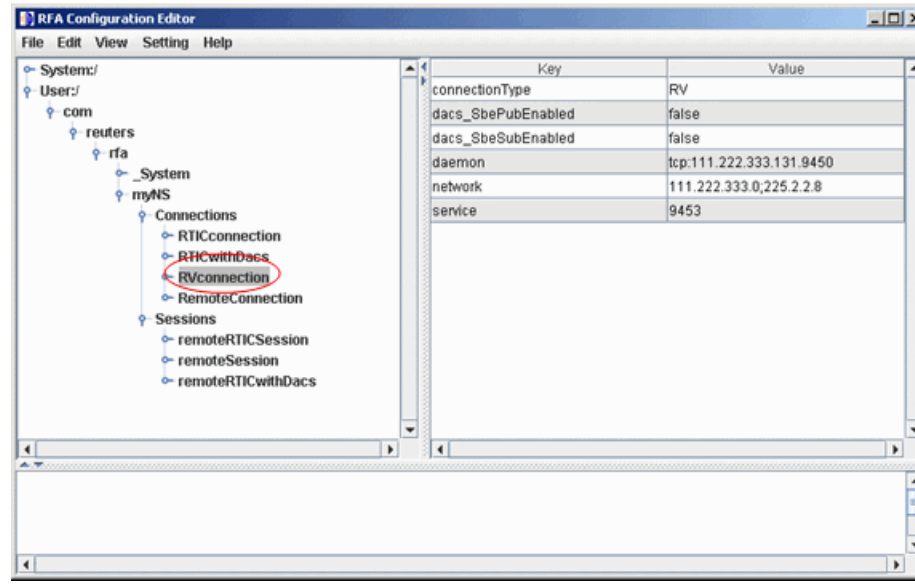
- **DACs-enabled**

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

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



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



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

```
Oct 5, 2007 2:27:14 PM ...
com.reuters.rfa.internal.connection.
ConnectionImpl$ConnectionEstablishmentThread runImpl
INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
Connection successful: ...
    componentName :myNS::RTICwithDacs,
subscriberRVConnection:
{service: 9453, network: 192.168.107.0;225.2.2.8,
daemon: tcp:192.168.107.131:9450}
Oct 5, 2007 2:27:14 PM
com.reuters.rfa.internal.connection.sass3....
    Sass3LoggerProxy log
INFO: com.reuters.rfa.connection.sass3.myNS.RTICwithDacs
SASS3JNI: Received advisory from RV session@
```

```
(9453,192.168.107.0;225.2.2.8,tcp:192.168.107.131:9450):
 _RV.INFO.SYSTEM.RVD.CONNECTED
Oct 5, 2007 2:27:14 PM
com.reuters.rfa.internal.connection.ConnectionImpl
makeServiceInfo
WARNING: com.reuters.rfa.connection.sass3....
    myNS.RTICwithDacs
Service list configuration has no
    alias defined for network
serviceName IDN_RDF
```

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

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

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

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

See Also

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

Purpose

Create Reuters Instrument Code

Syntax

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

Description

`addric(r,ric,fid,fval,type)` creates a Reuters Instrument Code, `ric`, on the service defined by the Reuters session, `r`. Supply the field ID or name, `fid`, and the field value, `fval`. Specify whether the RIC `type` is `'live'` or `'static'` (default).

Examples

Create a live RIC called `myric` with the fields `trdprc_1` (field ID 6) and `bid` (field ID 22) set to initial values of 0:

```
addric(r,'myric',{trdprc_1','bid'},{0,0},'live')
```

Create a live RIC called `myric` with the fields `trdprc_1` and `bid` set to initial values of 0:

```
addric(r,'myric',{6,22},{0,0},'live')
```

See Also

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

reuters.close

Purpose

Release connections to Reuters data servers

Syntax

```
close(r)
```

Arguments

`r`	Reuters session object created with the `reuters` function

Description

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

Examples

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

```
close(r)
```

See Also

`reuters`

Purpose

Contribute data to Reuters datafeed

Syntax

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

Description

`contrib(r,s,fid,fval)` contributes data to a Reuters datafeed. `r` is the Reuters session object, and `s` is the RIC. Supply the field IDs or names, `fid`, and field values, `fval`.

Examples

Contribute data to the Reuters datafeed for the Reuters session object `r` and the RIC `'myric'`. Provide a last trade price of 33.5.

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

Contribute an additional bid price of 33.8:

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

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

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

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

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

See Also

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

reuters.deleteric

Purpose

Delete Reuters Instrument Code

Syntax

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

Description

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

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

Examples

Delete `myric` and all of its fields:

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

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

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

See Also

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

Purpose

Request data from Reuters data servers

Syntax

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

Arguments

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

Description

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

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

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

Examples

Retrieving Current Securities Data

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

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

Following is a partial listing of the returned security data:

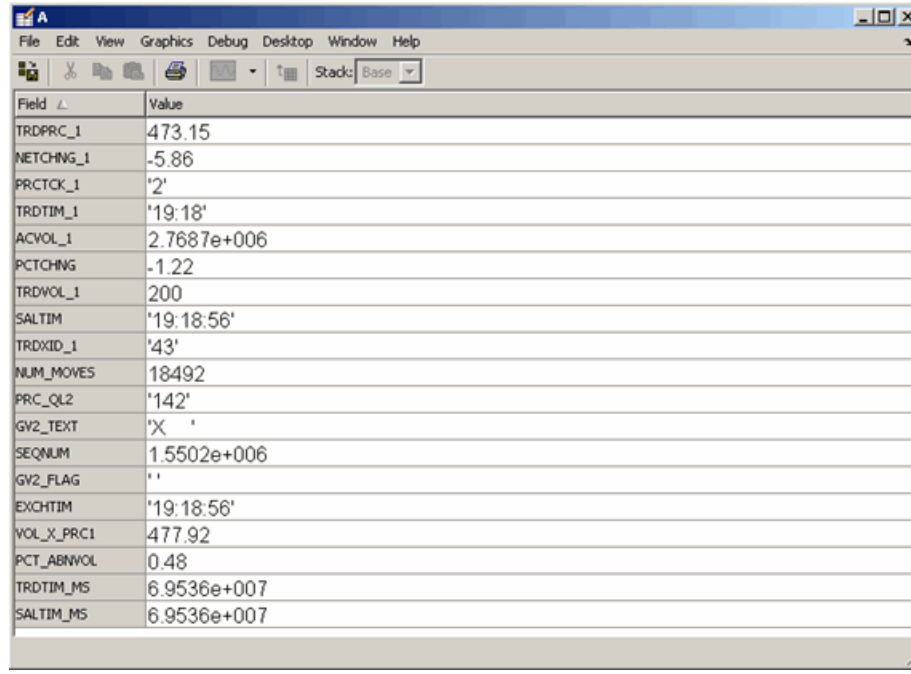
```
d =
PROD_PERM: 74.00
```

```
RDNDISPLAY: 66.00
DSPLY_NAME: 'DELAYED-15GOOGLE'
RDN_EXCHID: 'O'
TRDPRC_1: 474.28
TRDPRC_2: 474.26
TRDPRC_3: 474.25
TRDPRC_4: 474.25
TRDPRC_5: 474.25
NETCHNG_1: -4.73
HIGH_1: 481.35
LOW_1: 472.78
PRCTCK_1: '1'
CURRENCY: '840'
TRADE_DATE: '30 APR 2007
```

Subscribing to a Security

To subscribe to a security and process the data in real time, specify a `callback` function. MATLAB runs this function each time it receives a real-time data event from Reuters. In this example, the callback function, `rtdemo`, returns the subscription handle associated with this request to the base MATLAB workspace, `A`. The `openvar` function is then called to display `A` in the Variable Editor. A partial list of the data included in `A` appears in the figure.

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



Field	Value
TRDPRC_1	473.15
NETCHNG_1	-5.86
PRCTCK_1	'2'
TRDTIM_1	'19:18'
ACVOL_1	2.7687e+006
PCTCHNG	-1.22
TRDVOL_1	200
SALTIM	'19:18:56'
TRDXID_1	'43'
NUM_MOVES	18492
PRC_QL2	'142'
GV2_TEXT	'X '
SEQNUM	1.5502e+006
GV2_FLAG	' '
EXCHTIM	'19:18:56'
VOL_X_PRC1	477.92
PCT_ABNVOL	0.48
TRDTIM_MS	6.9536e+007
SALTIM_MS	6.9536e+007

See Also

`reuters` | `reuters.close` | `reuters.stop`

reuters.get

Purpose

Retrieve properties of Reuters session objects

Syntax

```
e = get(r)
e = get(r,f)
```

Arguments

`r`	Reuters session object created with the `reuters` function
`f`	Reuters session properties list

Description

`e = get(r)` returns Reuters session properties for the Reuters session object `r`.

`e = get(r,f)` returns Reuters session properties specified by the properties list `f` for the Reuters session object `r`.

See Also

`reuters`

Purpose

Request data from Reuters Time Series One

Syntax

```
d = history(r,s)
d = history(r,s,p)
d = history(r,s,f)
d = history(r,s,f,p)
d = history(r,s,d)
d = history(r,s,startdate,enddate)
d = history(r,s,startdate,enddate,p)
d = history(r,s,f,startdate,enddate)
d = history(r,s,f,startdate,enddate,p)
```

Description

`d = history(r,s)` returns all available daily historical data for the RIC, `s`, for the Reuters session object `r`.

`d = history(r,s,p)` returns all available historical data for the RIC, `s`, for the Reuters session object `r`. `p` specifies the period of the data,

- `'d'` - daily (default)
- `'w'` - weekly
- `'m'` - monthly

Note Reuters Time Series One will only return two years of daily data, five years of weekly data, or ten years of monthly data from the current date.

`d = history(r,s,f)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `r`.

`d = history(r,s,f,p)` returns all available historical data for the RIC, `s`, and fields, `f`, for the Reuters session object `r`. `p` specifies the period of the data.

`d = history(r,s,d)` returns the historical data for the RIC, `s`, for the given date, `d`, for the Reuters session object `r`.

`d = history(r,s,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(r,s,startdate,enddate,p)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

`d = history(r,s,f,startdate,enddate)` returns the daily historical data for the RIC, `s`, for the given date range defined by `startdate` and `enddate`.

`d = history(r,s,f,startdate,enddate,p)` returns the historical data for the RIC, `s`, and fields, `f`, for the given date range defined by `startdate` and `enddate`. `p` specifies the period of the data.

Examples

`d = history(r,'WXYZ.O')` returns a structure containing all available historical end of day daily data for the RIC `wxyz.o`, for the Reuters session object `r`.

`d = history(r,'WXYZ.O','close')` returns a structure with the fields `date` and `close` containing all available historical end of day daily data for the RIC `wxyz.o`.

`d = history(r,'WXYZ.O','close','m')` returns all available monthly data.

`d = history(r,'WXYZ.O','01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009. Note that only two years worth of daily data, five years worth of weekly data, and 10 years of monthly data from today's date is made available by Reuters.

`d = history(r,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009')` returns all available daily data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

```
d =
history(r,'WXYZ.O',{'close','volume'},'01-03-2009','02-24-2009','w')
```

returns all available weekly data for the date range 01-03-2009 to 02-24-2009 for the fields `date`, `close` and `volume`.

See Also

`reuters` | `reuters.fetch`

reuters.stop

Purpose

Unsubscribe securities

Syntax

```
stop(r)
stop(r,d)
```

Arguments

`r`	Reuters session object created with the `reuters` function
`d`	Subscription handle returned by `reuters/fetch`

Description

`stop(r)` unsubscribes all securities associated with the Reuters session object `r`.

`stop(r,d)` unsubscribes the securities associated with the subscription handle `d`, where `d` is the subscription handle returned by `reuters/fetch`.

Examples

Unsubscribe securities associated with a specific request `d` and a Reuters connection object `r`:

```
stop(r,d)
```

Unsubscribe all securities associated with the Reuters connection object `r`:

```
stop(r)
```

See Also

`reuters | reuters.fetch`

Purpose

Reuters Market Data System configuration editor

Syntax

```
rmdsconfig
```

Description

`rmdsconfig` opens the Reuters Market Data System configuration editor.

See Also

`reuters`

rnseloader

Purpose

Retrieve data from Reuters Newscope sentiment archive file

Syntax

```
x = rnseloader(file)
x = rnseloader(file, 'date', {DATE1})
x = rnseloader(file, 'date', {DATE1, DATE2})
x = rnseloader(file, 'security', {SECNAME})
x = rnseloader(file, 'start', STARTREC)
x = rnseloader(file, 'records', NUMRECORDS)
```

Arguments

Specify the following arguments as name-value pairs. You can specify any combination of name-value pairs in a single call to `rnseloader`.

`file`	Reuters Newscope sentiment archive file from which to retrieve data.
`'date'`	Use this argument with `{DATE1, DATE2}` to retrieve data between and including the specified dates. Specify the dates as numbers or strings.
`'security'`	Use this argument to retrieve data for `SECNAME`, where `SECNAME` is a cell array containing a list of security identifiers for which to retrieve data.
`'start'`	Use this argument to retrieve data beginning with the record `STARTREC`, where `STARTREC` is the record at which `rnseloader` begins to retrieve data. Specify `STARTREC` as a number.
`'records'`	Use this argument to retrieve `NUMRECORDS` number of records.

Description

`x = rnseloader(file)` retrieves data from the Reuters Newscope sentiment archive file `file`, and stores it in the structure `x`.

`x = rnseloader(file, 'date', {DATE1})` retrieves data from `file` with date stamps of value `DATE1`.

`x = rnseloader(file, 'date', {DATE1, DATE2})` retrieves data from `file` with date stamps between `DATE1` and `DATE2`.

`x = rnseloader(file, 'security', {SECNAME})` retrieves data from `file` for the securities specified by `SECNAME`.

`x = rnseloader(file, 'start', STARTREC)` retrieves data from `file` beginning with the record specified by `STARTREC`.

`x = rnseloader(file, 'records', NUMRECORDS)` retrieves `NUMRECORDS` number of records from `file`.

Examples

Retrieve data from the file `file.csv` with date stamps of `02/02/2007`:

```
x = rnseloader('file.csv','date',{'02/02/2007'})
```

Retrieve data from `file.csv` between and including `02/02/2007` and `02/03/2007`:

```
 x = rnseloader('file.csv','date',{'02/02/2007',...
'02/03/2007'})
```

Retrieve data from `file.csv` for the security `XYZ.O`:

```
 x = rnseloader('file.csv','security',{'XYZ.O'})
```

Retrieve the first 10000 records from `file.csv`:

```
 x = rnseloader('file.csv','records',10000)
```

Retrieve data from `file.csv`, starting at record 100000:

```
  x = rnseloader('file.csv','start',100000)
```

Retrieve up to 100000 records from `file.csv`, for the securities `ABC.N` and `XYZ.O` , with date stamps between and including the dates `02/02/2007` and `02/03/2007`:

```
x = rnseloader('file.csv','records',100000,...
               'date',{'02/02/2007','02/03/2007'},...
               'security',{'ABC.N','XYZ.O'})
```

See Also

`reuters` | `rdthloader`

Purpose

SIX Telekurs connection

Syntax

```
T = tlkrs(CI,UI,password)
```

Description

`T = tlkrs(CI,UI,password)` makes a connection to the SIX Telekurs data service given the CustomerID, `CI`, UserID, `UI`, and password, `password` provided by SIX Telekurs.

See Also

`tlkrs.close` | `tlkrs.getdata` | `tlkrs.history` | `tlkrs.timeseries`

tlkrs.close

Purpose

Close connection to SIX Telekurs

Syntax

```
close(C)
```

Description

`close(C)` closes the connection, `C`, to SIX Telekurs.

See Also

`tlkrs`

Purpose

Current SIX Telekurs 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 Telekurs 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` | `tlkrs.history` | `tlkrs.timeseries` | `tlkrs.tkfieldtoid` | `tlkrs.tkidtofield`

Purpose

End of day SIX Telekurs 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 Telekurs data for the specified security for the past 5 days.

```
c = tlkrs('US12345','userapid01','userapid10')
ids = tkfieldtoid(c,{'Bid','Ask'},'history');
d = history(c,{'1758999,149,134'},ids,floor(now)-5,floor(now));
```

```
d =

    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
     HL: [1x1 struct]
     HD: [1x1 struct]
      P: [1x1 struct]
```

`d.I` contains the instrument IDs, `d.HD` contains the dates, and `d.P` contains the pricing data.

View the dates:

```
d.HD.d
```

```
ans =

    '20110225'
    '20110228'
    '20110301'
```

View the pricing field IDs:

```
d.P.k

ans =

    '3,2'
    '3,3'
    '3,2'
    '3,3'
    '3,2'
    '3,3'
```

View the pricing data:

```
d.P.v

ans =

    '45.32'
    '45.33'
    '45.26'
    '45.27'
    '44.94'
    '44.95'
```

Convert the field identification strings in `d.P.k` to their corresponding field names like this:

```
d.P.k = tkidtofield(c,d.P.k,'history')
```

See Also

`tlkrs` | `tlkrs.getdata` | `tlkrs.timeseries` | `tlkrs.tkfieldtoid` | `tlkrs.tkidtofield`

Purpose

True if valid SIX Telekurs connection

Syntax

```
X = isconnection(C)
```

Description

`X = isconnection(C)` returns `true` if `C` is a valid SIX Telekurs connection and `false` otherwise.

See Also

`tlkrs` | `tlkrs.close` | `tlkrs.getdata`

tlkrs.timeseries

Purpose

SIX Telekurs 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 Telekurs connection object `c`, the security `s`, and the date `t`. Every trade, best, and ask tick is returned for the given date or date range.

`D = timeseries(c,s,{startdate,enddate})` returns the raw tick data for the security `s`, for the date range defined by `startdate` and `enddate`.

`D = timeseries(c,s,t,5)` returns the tick data for the security `s`, for the date `t` in intervals of 5 minutes, for the field `f`. Intraday tick data requested is returned in 5-minute intervals, with the columns representing First, High, Low, Last, Volume Weighted Average, and Moving Average.

Examples

Retrieve SIX Telekurs 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 the `tkidtofield` method:

```
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` | `tlkrs.getdata` | `tlkrs.history`

Purpose

SIX Telekurs field names to identification string

Syntax

```
D = tkfieldtoid(c,f,typ)
```

Description

`D = tkfieldtoid(c,f,typ)` converts SIX Telekurs field names to their corresponding identification strings. `c` is the SIX Telekurs 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 the `tlkrs.getdata` method, `tass` fields are used with the `tlkrs.timeseries` method, and `history` fields are used with the `tlkrs.history` method.

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` | `tlkrs.getdata` | `tlkrs.history` | `tlkrs.timeseries` | `tlkrs.tkidtofield`

tlkrs.tkidtofield

Purpose

SIX Telekurs identification string to field name

Syntax

```
D = tkidtofield(c,f,typ)
```

Description

`D = tkidtofield(c,f,typ)` converts SIX Telekurs field identification strings to their corresponding field names. `c` is the SIX Telekurs 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 the `tlkrs.getdata` method, `tass` fields are used with the `tlkrs.timeseries` method, and `history` fields are used with the `tlkrs.history` method.

Examples

When you retrieve output from SIX Telekurs, it appears as follows:

```
d =
    XRF: [1x1 struct]
     IL: [1x1 struct]
      I: [1x1 struct]
      M: [1x1 struct]
      P: [1x1 struct]
```

The instrument IDs are found in `d.I`, and the pricing data is found in `d.P`. The output for `d.P.k` appears like this:

```
ans =

    '33,2,1'
    '33,3,1'
    '3,1,1'
    '33,2,1'
    '33,3,1'
    '3,1,1'
```

Convert the field IDs in `d.P.k` to their field names with the `tkidtofield` method:

```
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` | `tlkrs.getdata` | `tlkrs.history` | `tlkrs.timeseries` | `tlkrs.tkfieldtoid`

xtrdr

Purpose

X_TRADER connection

Syntax

```
X = xtrdr
```

Description

`X = xtrdr` starts X_TRADER or connects to an existing X_TRADER session.

Output Arguments

`X`

X_TRADER connection.

Limitations

- You should only create one X_TRADER connection per MATLAB session. To create a new X_TRADER connection, start a new MATLAB session.

See Also

`xtrdr.close`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9
- "X_TRADER Order Submission" on page 2-13

Purpose

Terminate X_TRADER connection

Syntax

```
close(X)
```

Description

`close(X)` terminates the X_TRADER connection `X`.

Input Arguments

`X`

Connection to an X_TRADER session

See Also

`xtrdr`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9
- "X_TRADER Order Submission" on page 2-13

xtrdr.createInstrument

Purpose

Instruments for X_TRADER

Syntax

```
createInstrument(X,S)
createInstrument(X,Name,Value)
```

Description

`createInstrument(X,S)` creates the `xtrdr` instrument defined by the structure `S` with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`createInstrument(X,Name,Value)` creates the instrument using one or more `Name,Value` pair arguments with names and values corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Input Arguments

`X`

`xtrdr` connection object.

`S`

Input structure with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Examples

Create an xtrdr Instrument Using an Input Structure

Start X_TRADER.

```
X = xtrdr;
```

Define an input structure, `S`, with fields corresponding to valid X_TRADER API options.

```
S = [];
S.Exchange = 'Eurex';
S.Product = 'OGBM';
S.ProdType = 'Option';
```

```
S.Contract = 'Jan12 P12300';
S.Alias = 'TestInstrument3';
S

S =

    Exchange: 'Eurex'
     Product: 'OGBM'
    ProdType: 'Option'
    Contract: 'Jan12 P12300'
       Alias: 'TestInstrument3'
```

Create an `xtrdr` instrument.

```
createInstrument(X,S);
```

Close the connection.

```
close(X)
```

Create an xtrdr Instrument Using Name-Value Pairs

Start X_TRADER.

```
X = xtrdr;
```

Create an `xtrdr` instrument using name-value pairs corresponding to valid X_TRADER API options.

```
createInstrument(X,'Exchange','Eurex','Product','OGBM',...
                 'ProdType','Option','Contract','Jan12 P12300',...
                 'Alias','TestInstrument3');
```

Close the connection.

```
close(X)
```

See Also

`xtrdr` | `xtrdr.createNotifier` | `xtrdr.createOrderProfile` | `xtrdr.createOrderSet`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9
- "X_TRADER Order Submission" on page 2-13

Purpose

Instrument notifier for X_TRADER

Syntax

```
createNotifier(X,S)
createNotifier(X,Name,Value)
```

Description

`createNotifier(X,S)` creates the `xtrdr` instrument notifier defined by the structure `S` with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`createNotifier(X,Name,Value)` creates the instrument notifier using X_TRADER API options specified by one or more `Name,Value` pair arguments with names and values corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Input Arguments

`X`

`xtrdr` connection object.

`S`

Input structure with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Examples

Create an xtrdr Instrument Notifier Using an Input Structure

Start X_TRADER.

```
X = xtrdr;
```

Define an input structure, `S`, with fields corresponding to valid X_TRADER API options.

```
S = [];
S.Instrument = [];
S.UpdateFilter = '';
S.EnablePriceUpdates = -1;
```

```
S.EnableDepthUpdates = 0;
S.DebugLogLevel = 3;
S.EnableOrderSetUpdates = -1;
S.PriceList = [];
S.DeliverAllPriceUpdates = 0;
S
```

```
S =

               Instrument: []
             UpdateFilter: ''
       EnablePriceUpdates: -1
       EnableDepthUpdates: 0
            DebugLogLevel: 3
    EnableOrderSetUpdates: -1
                PriceList: []
   DeliverAllPriceUpdates: 0
```

Create an `xtrdr` instrument notifier.

```
createNotifier(X,S);
```

Close the connection.

```
close(X)
```

Create an xtrdr Instrument Notifier Using Name-Value Pairs

Start X_TRADER.

```
X = xtrdr;
```

Create an `xtrdr` instrument using name-value pairs corresponding to valid X_TRADER API options.

```
createNotifier(X,'Instrument',[],'UpdateFilter','',...
               'EnablePriceUpdates',-1,'EnableDepthUpdates',0,...
               'DebugLogLevel',3,'EnableOrderSetUpdates',-1,...
               'PriceList',[],'DeliverAllPriceUpdates',0);
```

Close the connection.

```
close(X)
```

See Also

`xtrdr | xtrdr.createInstrument | xtrdr.createOrderProfile | xtrdr.createOrderSet`

Related Examples

- "X_TRADER Price Update" on page 2-7
- "X_TRADER Price Update Depth" on page 2-9
- "X_TRADER Order Submission" on page 2-13

xtrdr.createOrderProfile

Purpose

Order profiles for X_TRADER

Syntax

```
P = createOrderProfile(X,S)
P = createOrderProfile(X,Name,Value)
```

Description

`P = createOrderProfile(X,S)` creates an order profile defined by the structure `S` with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`P = createOrderProfile(X,Name,Value)` creates an order profile using X_TRADER API options specified by one or more `Name,Value` pair arguments with names and values corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Input Arguments

`X`

`xtrdr` connection object.

`S`

Input structure with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Output Arguments

`P`

Order profile.

Examples

Create an Order Profile Using an Input Structure

Start X_TRADER.

```
X = xtrdr;
```

Define an input structure, `S`, with fields corresponding to valid X_TRADER API options.

```
S = [];
S.Instrument = [];
S.Customer = '';
S.Alias = '';
S.ReadProperties = 'b';
S.WriteProperties = 'b';
S.Customers = {'<Default>'};
S.RoundOption = 2;
S.CustomerDefaults = [];
S
```

```
S =

          Instrument: []
            Customer: ''
               Alias: ''
      ReadProperties: 'b'
     WriteProperties: 'b'
           Customers: {'<Default>'}
         RoundOption: 2
    CustomerDefaults: []
```

Create an order profile.

```
P = createOrderProfile(X,S);
```

Close the connection.

```
close(X)
```

Create an Order Profile Using Name-Value Pairs

Start X_TRADER.

```
X = xtrdr;
```

Create an order profile using name-value pairs corresponding to valid X_TRADER API options.

```
createOrderProfile(X,'Instrument',[],'Customer','',...
             'Alias','','ReadProperties','b',...
             'WriteProperties','b','Customers',{'<Default>'},...
             'RoundOption',2,'CustomerDefaults',[]);
```

Close the connection.

```
close(X)
```

See Also

xtrdr | xtrdr.createInstrument | xtrdr.createNotifier | xtrdr.CreateOrderSet

Related Examples

- “X_TRADER Order Submission” on page 2-13

Purpose

Order set for X_TRADER

Syntax

```
createOrderSet(X,S)
createOrderSet(X,Name,Value)
```

Description

`createOrderSet(X,S)` creates an `xtrdr` order set defined by the structure `S` with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`createOrderSet(X,Name,Value)` creates an order set using X_TRADER API options specified by one or more `Name,Value` pair arguments with names and values corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Input Arguments

`X`

`xtrdr` connection object.

`S`

Input structure with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

Examples

Create an Order Set Using an Input Structure

Start X_TRADER.

```
X = xtrdr;
```

Define an input structure, `S`, with fields corresponding to valid X_TRADER API options.

```
S = [];
S.Count = 0;
S.Alias = '';
S.ReadProperties = 'b';
```

```
S.WriteProperties = 'b';
S.EnableOrderSetUpdates = -1;
S.EnableOrderFillData = 0;
S.EnableOrderSend = 0;
S.EnableOrderAutoDelete = 0;
S.QuotingOrderProfile = [];
S.DebugLogLevel = 3;
S.QuoteWithCancelReplace = 0;
S.EnableOrderUpdateData = 0;
S.EnableFillCaching = 0;
S.AvgOpenPriceMode = 'NONE';
S.EnableOrderRejectData = 0;
S.OrderStatusNotifyMode = 'ORD_NOTIFY_NONE';
```

Create an order set.

```
createOrderSet(X,S);
```

Close the connection.

```
close(X)
```

Create an Order Set Using Name-Value Pairs

Start X_TRADER.

```
X = xtrdr;
```

Create an order set using name-value pairs corresponding to valid X_TRADER API options.

```
createOrderSet(X,'Count',0,'Alias','','ReadProperties','b',...
               'WriteProperties','b','EnableOrderSetUpdates',-1,...
               'EnableOrderFillData',0,'EnableOrderSend',0,...
               'EnableOrderAutoDelete',0,'QuotingOrderProfile',[],...
               'DebugLogLevel,3,'QuoteWithCancelReplace',0,...
               'EnableOrderUpdateData',0,'EnableFillCaching',0,...
               'AvgOpenPriceMode','NONE','EnableOrderRejectData',0,...
               'OrderStatusNotifyMode','ORD_NOTIFY_NONE');
```

Close the connection.

```
close(X)
```

See Also

`xtrdr` | `xtrdr.createInstrument` | `xtrdr.createNotifier` | `xtrdr.createOrderProfile`

Related Examples

- "X_TRADER Order Submission" on page 2-13

xtrdr.getData

Purpose

Current X_TRADER data

Syntax

```
D = getData(X,S,F)
D = getData(X,S,F)
D = getData(X,F)
```

Description

`D = getData(X,S,F)` returns data for the fields `F` for the `xtrdr` instrument object, `S`, with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`D = getData(X,S,F)` returns data for the fields `F` for the `xtrdr` instrument aliases, `S`.

`D = getData(X,F)` returns data for the fields `F` for all instruments associated with the `xtrdr` session object, `X`.

Input Arguments

`X`

`xtrdr` connection object.

`S`

Instrument object or aliases with fields corresponding to valid X_TRADER API options. For details, see the Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

`F`

Fields for the instrument object or aliases, `S`. `F` without a corresponding `S` are fields for all instruments associated with the `xtrdr` session object, `X`.

Output Arguments

`D`

X_TRADER data.

Examples

Return Exchange and Last Price for an Instrument

Return the exchange and last price fields for the instrument defined in `x.Instrument(1)`.

```
D = getData(X,X.Instrument(1),{'Exchange','Last'});
```

Return Exchange and Last Price for an Alias

Return the exchange and last price fields for the instrument defined by the alias `PriceInstrument1`.

```
D = getData(X,'PriceInstrument1',{'Exchange','Last'});
```

Return Exchange and Last Price for All Session Instruments

Return the exchange and last price fields for all instruments associated with the `xtrdr` session object, `X`.

```
D = getData(X,{'Exchange','Last'});
```

See Also

`xtrdr | xtrdr.createInstrument`

Related Examples

- “X_TRADER Price Update Depth” on page 2-9

yahoo

Purpose

Connect to Yahoo! data servers

Syntax

```
Connect = yahoo
```

Description

`Connect = yahoo` verifies that the URL `http://quote.yahoo.com` is accessible and creates a connection handle.

Examples

Connect to the Yahoo! data server:

```
Connect = yahoo
Connect =
     url: 'http://quote.yahoo.com'
```

See Also

`yahoo.builduniverse` | `yahoo.close` | `yahoo.fetch` | `yahoo.get` | `yahoo.isconnection` | `yahoo.trpdata`

Purpose

Portfolio matrix with total return price data from Yahoo!

Syntax

```
X = builduniverse(y,s,d1,d2,p)
```

Description

`X = builduniverse(y,s,d1,d2,p)` builds a portfolio matrix using Yahoo! data to compute a total return price series. `X` is an m-by-(n+1) matrix, where m refers to the number of records of data and n refers to the number of securities. Column 1 of the matrix contains MATLAB date numbers and the remaining columns are the total return prices for each security. `y` is the Yahoo! connection handle, `s` is a cell array of security identifiers, `d1` and `d2` are the start and end dates for the data request, and `p` is the periodicity flag. `p` can be entered as:

- `'d'` for daily values
- `'w'` for weekly values
- `'m'` for monthly values

Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by this method contains the expected results.

Examples

Compute a total return price series and convert to daily total returns:

```
y = yahoo;
%
% Load security list.
s = {'A', 'B', 'C'};

% Get a daily total return price series for securities.
% (Calculated from prices, splits and dividends.)
Universe = builduniverse(y,s,'1/15/2007',floor(now));

% Convert to daily total returns.
Universe = periodicreturns(Universe,'d');
```

See Also

`yahoo.fetch` | `yahoo.trpdata`

yahoo.close

Purpose

Close connections to Yahoo! data servers

Syntax

```
close(Connect)
```

Arguments

`Connect`	Yahoo! connection object created with the `yahoo` function.

Description

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

See Also

`yahoo`

Purpose

Request data from Yahoo! data servers

Syntax

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

Arguments

`Connect`	Yahoo! connection object created with the `yahoo` function.
`Security`	A MATLAB string or cell array of strings containing the name of a security 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 only a single security at a time.

`Fields` A MATLAB string or cell array of strings indicating the data fields for which to retrieve data. A partial list of supported values for current market data are:

- `'Symbol'`
- `'Last'`
- `'Date'`
- `'Time'`

> **Note** `'Date'` and `'Time'` are MATLAB date numbers. (`'Time'` is a fractional part of a date number. For example, 0.5 = 12:00:00 PM.)

- `'Change'`
- `'Open'`
- `'High'`
- `'Low'`
- `'Volume'`

A partial list of supported values for historical data are:

- `'Close'`
- `'Date'`
- `'High'`
- `'Low'`
- `'Open'`
- `'Volume'`
- `'Adj Close'`

	For a complete list of supported values for market and historical data, see `yhfields.mat`.
`Date`	Date string or serial date number indicating date for the requested data. If you enter today's date, `fetch` returns yesterday's data.
`FromDate`	Beginning date for historical data.
	Note You can specify dates in any of the formats supported by `datestr` and `datenum` that show a year, month, and day.
`ToDate`	End date for historical data.
`Period`	Period within date range. `Period` values are: • `'d'`: daily • `'w'`: weekly • `'m'`: monthly • `'v'`: dividends

Description

`data = fetch(Connect, 'Security')` returns data for all fields from Yahoo!'s 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.

`data = fetch(Connect, 'Security', 'Fields')` returns data for the specified fields.

`data = fetch(Connect, 'Security', 'Date')` returns all security data for the requested date.

`data = fetch(Connect, 'Security', 'Fields', 'Date')` returns security data for the specified fields on the requested date.

`data = fetch(Connect, 'Security', 'FromDate', 'ToDate')` returns security data for the date range `FromDate` to `ToDate`.

`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 indicated period.

Examples

Retrieving Last Prices for a Set of Equities

Connect to the Yahoo! data server to obtain the last prices for a set of equities:

```
y = yahoo;
FastFood = fetch(y, {'ko', 'pep', 'mcd'},'Last')
FastFood =
    Last: [3x1 double]
FastFood.Last
ans =
         42.96
         45.71
         23.70
```

Retrieving a Closing Price on a Specified Date

Obtain the closing price for Coca-Cola on April 6, 2010:

```
c = yahoo;
ClosePrice = fetch(c,'ko','Close','Apr 6 2010')
ClosePrice =
  1.0e+005 *
    7.3058    0.0005
```

See Also yahoo.close | yahoo.get | yahoo.isconnection | yahoo

yahoo.get

Purpose

Retrieve properties of Yahoo! connection objects

Syntax

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

Arguments

`Connect`	Yahoo! connection object created with the `yahoo` function.
`PropertyName`	(Optional) A MATLAB string or cell array of strings containing property names. Currently the only property name recognized is `'url'`.

Description

`value = get(Connect, 'PropertyName')` returns the value of the specified properties for the Yahoo! connection object.

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

Examples

Connect to a Yahoo! data server:

```
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

yahoo.close | yahoo.fetch | yahoo.isconnection | yahoo

yahoo.isconnection

Purpose

Verify whether connections to Yahoo! data servers are valid

Syntax

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

Arguments

`Connect`	Yahoo! connection object created with the `yahoo` function.

Description

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

Examples

Connect to a Yahoo! data server:

```
c = yahoo
```

Verify that the connection, `c`, is valid:

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

See Also

`yahoo.close` | `yahoo.fetch` | `yahoo.get` | `yahoo`

Purpose

Total return price series data

Syntax

```
[prc,act,div] = trpdata(y,s,d1,d2,p)
```

Description

`[prc,act,div] = trpdata(y,s,d1,d2,p)`, where `y` is the Yahoo! connection handle, `s` is the security string, `d1` is the start date, `d2` is the end date, and `p` is the periodicity flag for Yahoo!, generates a total return price series. `prc` is the price, `act` is the action, and `div` is the dividend returned in the total return price series.

Tips

- Data providers report price, action, and dividend data differently. Verify that the data returned by this method contains the expected results.

A

Examples

Use this list to find examples in the documentation.

Retrieving Connection Properties

"Connection Object Properties" on page 2-4
"Example: Retrieve Data on a Security" on page 2-5

Index

B

Bloomberg®
  connection object 2-3
Bloomberg® 6-2

C

close 6-115
  Bloomberg® 6-3
  FactSet® 6-67
  FRED® 6-75
  Haver Analytics® 6-82
  Interactive Data Pricing and Reference Data's RemotePlus™ 6-94
  Kx Systems®, Inc. 6-101
  Reuters® 6-142
  Thomson Reuters™ Datastream® 6-52
  Yahoo!® 6-188
connection object 2-3
CUSIP number 6-7

D

data servers
  disconnecting from 2-6
  retrieving connection properties 2-4
data services
  connection requirements 1-4
    for FactSet® data server 1-5
    for Reuters® data server 1-5
    for Thomson Reuters™ Datastream® data server 1-8
    proxy information 1-4
    software 1-4
  data service providers 1-3
Datafeed Dialog Box
  starting 4-3
datastream 6-51
dftool 4-3
disconnecting from data servers 2-6

E

eSignal® 6-59
exec
  Kx Systems®, Inc. 6-102

F

factset 6-66
Federal Reserve Economic Data (FRED®) 6-74
fetch 6-116
  Bloomberg® 6-4
  FactSet® 6-68
  FRED® 6-76
  Haver Analytics® 6-83
  Interactive Data Pricing and Reference Data's RemotePlus™ 6-95
  Kx Systems®, Inc. 6-104
  Reuters® 6-145
  Thomson Reuters™ Datastream® 6-53
  Yahoo!® 6-189
fred 6-74
functions
  Bloomberg®
    bloomberg 6-2
    close 6-3
    fetch 6-4
    get 6-12
    getdata 6-14
    history 6-15
    isconnection 6-18
    isfield 6-19
    lookup 6-20
    pricevol 6-22
    realtime 6-21
    showtrades 6-24
    stockticker 6-26
    stop 6-28
    timeseries 6-30
  esig 6-59
  FactSet®

close 6-67
factset 6-66
fetch 6-68
get 6-71
isconnection 6-73
FRED®
close 6-75
fetch 6-76
fred 6-74
get 6-78
isconnection 6-79
Haver Analytics®
aggregation 6-81
close 6-82
fetch 6-83
get 6-85
haver 6-80
havertool 6-91
info 6-86
isconnection 6-88
nextinfo 6-89
Interactive Data Pricing and Reference Data's RemotePlus™
close 6-94
fetch 6-95
get 6-97
idc 6-93
isconnection 6-98
Kx Systems®, Inc.
close 6-101
exec 6-102
fetch 6-104
get 6-106
insert 6-107
isconnection 6-108
kx 6-99
tables 6-109
Reuters®
close 6-142
fetch 6-145
get 6-148
history 6-149
reuters 6-130
stop 6-152
Reuters® Datascope Tick History
fetch 6-116
rdth 6-110
rdth.close 6-115
rdth.get 6-120
rdth.isconnection 6-123
rdthloader 6-127
rnseloader 6-154
Thomson Reuters™ Datastream®
close 6-52
datastream 6-51
fetch 6-53
get 6-57
isconnection 6-58
Yahoo!®
close 6-188
fetch 6-189
get 6-194
isconnection 6-196
yahoo 6-186

G

get 6-120
Bloomberg® 6-12
FactSet® 6-71
FRED® 6-78
Haver Analytics® 6-85
Interactive Data Pricing and Reference Data's RemotePlus™ 6-97
Kx Systems®, Inc. 6-106
Reuters® 6-148
Thomson Reuters™ Datastream® 6-57
Yahoo!® 6-194
getdata
Bloomberg® 6-14

graphical user interface 4-1

H

haver 6-80 to 6-81
Haver Analytics® 6-80 to 6-81
havertool 6-91
    Haver Analytics® 6-91
history
    Bloomberg® 6-15
    Reuters® 6-149

I

idc 6-93
info
    Haver Analytics® 6-86
insert
    Kx Systems®, Inc. 6-107
isconnection 6-123
    Bloomberg® 6-18
    FactSet® 6-73
    FRED® 6-79
    Haver Analytics® 6-88
    Interactive Data Pricing and Reference Data's RemotePlus™ 6-98
    Kx Systems®, Inc. 6-108
    Thomson Reuters™ Datastream® 6-58
    Yahoo!® 6-196
isfield
    Bloomberg® 6-19

K

kx 6-99

L

lookup
    Bloomberg® 6-20

N

nextinfo
    Haver Analytics® 6-89

P

pricevol
    Bloomberg® 6-22

R

rdth 6-110
rdthloader 6-127
realtime
    Bloomberg® 6-21
retrieving connection properties 2-4
reuters 6-130
Reuters®
    Reuters® Configuration Editor 1-5
Reuters® Datascope Tick History file. *See* rdth.
    See rdthloader
Reuters® Newscope. *See* rnseloader
rnseloader 6-154

S

Securities Lookup dialog box 4-9
showtrades
    Bloomberg® 6-24
stockticker
    Bloomberg® 6-26
stop
    Bloomberg® 6-28
    Reuters® 6-152

T

tables
    Kx Systems®, Inc. 6-109
timeseries
    Bloomberg® 6-30

Y

yahoo 6-186