# Trading Toolbox™

## User's Guide

**R2013a**

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.

*Trading Toolbox™ User's Guide*




### Revision History

| | | |
|---|---|---|
| March 2013 | Online only | Version 1.0 (Release 2013a) |

# Contents

## 1 Getting Started



## 2 Communicate with Financial Order Data Servers



## 3 Workflow Models

## 4 Sample Code for Workflows



## 5 Functions — Alphabetical List

## Index

1

# Getting Started

# Product Description

### Access prices and send orders to trading systems

Trading Toolbox™ provides functions for accessing trade and quote pricing data, defining order types, and sending orders to financial trading markets. The toolbox lets you integrate streaming and event-based data into MATLAB®, enabling you to develop financial trading strategies and algorithms that analyze and react to the market in real time. You can build algorithmic or automated trading strategies that work across multiple asset classes, instrument types, and trading markets while integrating with industry-standard trade execution platforms.

With Trading Toolbox, you can subscribe to streams of tradable instrument data, including quotes, volumes, trades, market depth, and instrument metadata. You also can define order types and instructions for how to route and fill orders. Supported trading platforms for order execution include Bloomberg® EMSX and Trading Technologies® X_TRADER®.

## Key Features

- Access to current, intraday, event-based, and real-time tradable instrument data
- Data filtering by instrument and exchange
- Definable order types and execution instructions
- Bloomberg EMSX order execution
- Trading Technologies X_TRADER instrument pricing and order execution

# Trading System Providers

**In this section...**



## Supported Providers

This toolbox supports connections to financial trading systems provided by the following corporations:

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

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

- X_TRADER from Trading Technologies (`http://www.tradingtechnologies.com`)

See the MathWorks® Web site for the system requirements for connecting to these trading systems.

## Connection Requirements

To connect to some of these trading systems, additional requirements apply. The following data service providers require you to install proprietary software on your PC:

- Bloomberg EMSX

  **Note** You need Bloomberg Desktop software license for the host on which Trading Toolbox and MATLAB software are running.

- Trading Technologies X_TRADER

You must have a valid license for required client software on your machine.

For more information about how to obtain required software, contact your trading system sales representative.

2

# Communicate with Financial Order Data Servers

# Bloomberg EMSX Order Management

**In this section...**



## Connect to Bloomberg EMSX

This example shows how to use the `emsx` function to connect to Bloomberg EMSX.

1. If you haven't used the `emsx` function before, then add the file `blpapi3.jar` to the MATLAB Java® classpath. Use the `javaaddpath` function or edit your `classpath.txt` file.

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

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

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

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

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

2. Connect to the Bloomberg EMSX data server.

```
C = emsx(servicename)
```

You are now connected to the Bloomberg EMSX data server. Your output appears as follows:

```
C =

 emsx with properties:

      Session:
      Service:
    Ipaddress:
         Port:
```

`servicename` is a string. The available services are:

- Bloomberg EMSX test service is `'//blp/emapisvc_beta'`
- Bloomberg EMSX production service is `'//bmp/emapisvc'`

When you create a Bloomberg EMSX connection using `emsx`, the connection object properties are returned.

```
 C = emsx('//blp/emapisvc_beta')
```

```
C =

  emsx with properties:

      Session: [1x1 com.bloomberglp.blpapi.Session]
      Service: [1x1 com.bloomberglp.blpapi.impl.aQ]
    Ipaddress: 'localhost'
         Port: 8194
```

## Workflow for Bloomberg EMSX

The workflow for Bloomberg EMSX is versatile with many options for alternate flows in the process of creating, routing, and managing the status of an open order until it is filled.

1. Connect to Bloomberg EMSX using `emsx`.
2. Subscribe to orders and routes to obtain events on subsequent requests to Bloomberg EMSX for orders and routes.

Use the`orders` and `routes` functions.

3 Create a Bloomberg EMSX order.

Options in the flow of creating an order are:

- Create an order using `createOrder`.
- Create an order and route using `createOrderAndRoute`. Or get route information using `getRouteInfo` and then create an order and route using `createOrderAndRoute`.
- Create an order and route that uses a strategy with `createOrderAndRouteWithStrat`.

4 Modify an order, or modify the route.

Options in the flow of modifying an order are:

- Modify an order using `modifyOrder`.
- Modify a route with a strategy using `modifyRouteWithStrat`.
- Modify a route using `modifyRoute`.

5 Delete the order or delete a route.

Options in the flow of deleting an order are:

- Delete the order using `deleteOrder`.
- Delete a route using `deleteRoute`.

6 Manage open order status.

Options in the flow of managing order status are:

- Obtain order information using `getOrderInfo`.
- Obtain route information using `getRouteInfo`.
- Obtain broker information using `getBrokerInfo`.

7 Close the Bloomberg EMSX connection using `close`.

## Disconnect from Bloomberg EMSX

To close a data server connection and disconnect, use the `close` function for Bloomberg EMSX:

```
close(C)
```

You must have previously created the connection object using `emsx`.

# Trading Technologies X_TRADER Order Management

**In this section...**



## Connect to Trading Technologies X_TRADER

This example shows how to use the `xtrdr` function to connect to Trading Technologies X_TRADER.

Connect to X_TRADER.

```
X = xtrdr
```

The connection object properties appear as follows:

```
X =

  xtrdr with properties:

           Gate: [1x1 COM.Xtapi_TTGate_1]
    InstrNotify: []
     Instrument: []
       OrderSet: []
```

As you use the X_TRADER functions to create an instrument (`createInstrument`), define an instrument notifier (`createNotifier`), and submit an order set (`createOrderSet`), the `xtrdr` connection properties are updated.

## Workflows for Trading Technologies X_TRADER

You can use X_TRADER to monitor market price information and submit orders.

To monitor market price information:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Optionally, use `getData` to return information about the instrument that you have created.
4. Close the Trading Technologies X_TRADER connection using `close`.

To submit orders to X_TRADER:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Optionally, use `getData` to return information about the instrument that you have created.
4. Create an order set using `createOrderSet` to define the level of the order status events and event handlers for orders that will be submitted to X_TRADER.
5. Define the order using `createOrderProfile`. An order profile contains the settings that define an individual order to be submitted.
6. Route the order for execution using the `OrderSet` object created by `createOrderSet` in step 4.
7. Close the Trading Technologies X_TRADER connection using `close`.

To monitor market price information and respond to market changes by automatically submitting orders to X_TRADER:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Use `getData` to return information on the instrument that you have created.

4. Define events by assigning callbacks for validating or invalidating an instrument and performing calculations based on the event. Based on some predefined condition reached when changes in the incoming data satisfy the condition, event callbacks execute the functions in steps 5, 6, and 7.

5. Create an order set using `createOrderSet` to define the level of the order status events and event handlers for orders that will be submitted to X_TRADER.

6. Define the order using `createOrderProfile`. An order profile contains the settings that define an individual order for submission.

7. Route the order for execution using the `OrderSet` object created by `createOrderSet` in step 5.

8. Close the Trading Technologies X_TRADER connection using `close`.

## Disconnect from Trading Technologies X_TRADER

Use the `close` function for Trading Technologies X_TRADER:

```
close(X)
```

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

3

# Workflow Models

# Workflow for Bloomberg EMSX

The workflow for Bloomberg EMSX is versatile with many options for alternate flows to create, route, and manage the status of an open order until it is filled.

1. Connect to Bloomberg EMSX using `emsx`.
2. Subscribe to orders and routes to obtain events on subsequent requests to Bloomberg EMSX for orders and routes.

   Use the `orders` and `routes` functions.
3. Create a Bloomberg EMSX order. Options in the flow of creating an order are:
   - Create an order using `createOrder`.
   - Create an order and route using `createOrderAndRoute`. Or get route information using `getRouteInfo` and then create an order and route using `createOrderAndRoute`.
   - Create an order and route that uses a strategy with `createOrderAndRouteWithStrat`.
4. Modify an order, or modify the route. Options in the flow of modifying an order are:
   - Modify an order using `modifyOrder`.
   - Modify a route with a strategy using `modifyRouteWithStrat`.
   - Modify a route using `modifyRoute`.
5. Delete the order, or delete a route. Options in the flow of deleting an order are:
   - Delete the order using `deleteOrder`.
   - Delete a route using `deleteRoute`.
6. Manage open order status. Options in the flow of managing order status are:
   - Obtain order information using `getOrderInfo`.

- Obtain route information using `getRouteInfo`.
- Obtain broker information using `getBrokerInfo`.

**7** Close the Bloomberg EMSX connection using `close`.

## Related Examples


- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

# Workflows for Trading Technologies X_TRADER

You can use X_TRADER to monitor market price information and submit orders.

To monitor market price information:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Optionally, use `getData` to return information on the instrument that you have created.
4. Close the Trading Technologies X_TRADER connection using `close`.

To submit orders to X_TRADER:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Optionally, use `getData` to return information on the instrument that you have created.
4. Create an order set using `createOrderSet` to define the level of the order status events and event handlers for orders that will be submitted to X_TRADER.
5. Define the order using `createOrderProfile`. An order profile contains the settings that define an individual order to be submitted.
6. Route the order for execution using the `OrderSet` object created by `createOrderSet` in step 4.
7. Close the Trading Technologies X_TRADER connection using `close`.

To monitor market price information and respond to market changes by automatically submitting orders to X_TRADER:

1. Connect to Trading Technologies X_TRADER using `xtrdr`.
2. Create an event notifier using `createNotifier`.
3. Create an instrument and attach it to the notifier using `createInstrument`. Use `getData` to return information on the instrument that you have created.
4. Define events by assigning callbacks for validating or invalidating an instrument and performing calculations based on the event. Based on some predefined condition reached when changes in the incoming data satisfy the condition, event callbacks execute the functions in steps 5, 6, and 7.
5. Create an order set using `createOrderSet` to define the level of the order status events and event handlers for orders that will be submitted to X_TRADER.
6. Define the order using `createOrderProfile`. An order profile contains the settings that define an individual order to be submitted.
7. Route the order for execution using the `OrderSet` object created by `createOrderSet` in step 5.
8. Close the Trading Technologies X_TRADER connection using `close`.

## Related Examples


- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

4

# Sample Code for Workflows

# X_TRADER Workflows

X_TRADER supports the following workflows:

# X_TRADER Price Update

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

**Connect to X_TRADER.**

```
X = xtrdr;
```

**Create an event notifier.**

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

```
createNotifier(X);
```

**Create an instrument.**

Create an instrument and attach it to the notifier.

```
createInstrument(X,'Exchange','CME','Product','2F',...
                'ProdType','Future','Contract','Aug13',...
                '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 the desired fields. In this example, events are monitored for updates to last price, last quantity, previous last quantity, and a change in prices. Listen for this 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(X)
```

## See Also

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

## Related Examples

- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

# X_TRADER Price Update Depth

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

**Connect to X_TRADER.**

```
X = xtrdr;
```

**Create an event notifier.**

Create an event notifier and enable depth updates. The event notifier is the X_TRADER mechanism lets you define MATLAB functions to use as callbacks for specific events.

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

**Create an instrument.**

```
createInstrument(X,'Exhange','CME','Product','2F','ProdType','Future',...
               'Contract','Aug13','Alias','PriceInstrumentDepthUpdate');
```

**Attach an instrument to a notifier.**

Assign one or more notifiers to an instrument. A notifier can have one or more instruments attached to it.

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

**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]);
```

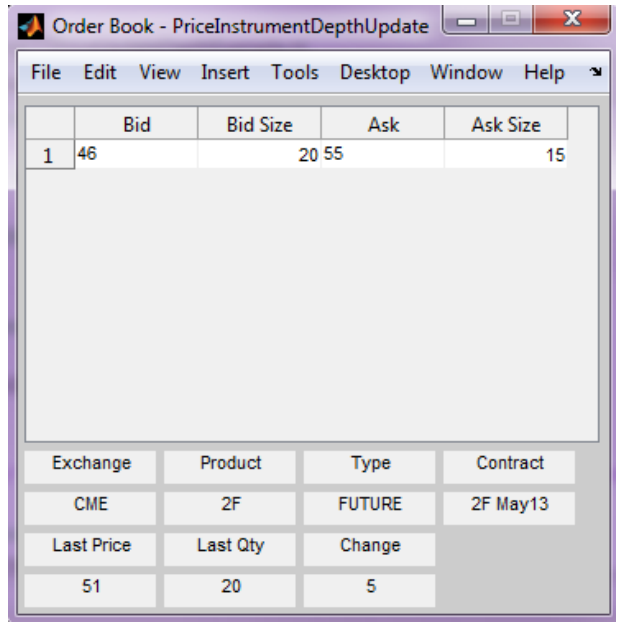
**Store data.**

```
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 instructs X_TRADER to start monitoring the attached instruments using the specified event settings.

**Close the connection.**

```
close(X)
```

## See Also

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

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

# X_TRADER Order Submission

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

**Connect to X_TRADER.**

```
X = xtrdr;
```

**Create an instrument.**

```
createInstrument(X,'Exchange','CME','Product','2F',...
                'ProdType','Future','Contract','Aug13',...
                'Alias','SubmitOrderInstrument1');
```

**Register event handlers.**

Register event handlers for the order server. The callback `ttorderserverstatus` is assigned to the event `OnExchangeStateUpdate` 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 sends 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 order submission.**

You must first enable order submission before you can 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:

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

**Sample: Create a market order.**

Create a market order to buy 100 shares.

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

**Sample: Create a limit order.**

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

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

**Sample: Create a stop market order.**

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

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

**Sample: Create a stop limit order.**

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

```
orderProfile.Set('OrderType','L');
orderProfile.Set('OrderRestr','S');
orderProfile.Set('Limit$','128000');
orderProfile.Set('Stop$','127500');
```

**Check the order server status.**

Check the order server status before submitting the order and add a counter so the example doesn't delay.

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

**Verify the 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(X)
```

## See Also

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

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

# Bloomberg EMSX Workflows

Bloomberg EMSX supports the following workflows:


- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

# Bloomberg EMSX Order Management

This example shows how to connect to a Bloomberg EMSX service, set up an order subscription, and create and manage an order.

**Connect to Bloomberg EMSX.**

```
b = emsx('//blp/emapisvc_beta');
processEvent(b)

SessionConnectionUp = {
    server = localhost/127.0.0.1:8194
}

SessionStarted = {
}

ServiceOpened = {
    serviceName = //blp/emapisvc_beta
}
```

**Set up the order subscription.**

```
r = b.orders({'EMSX_TICKER','EMSX_AMOUNT','EMSX_FILL'})

r =

                      MSG_TYPE: {'E'}
                  MSG_SUB_TYPE: {'O'}
                  EVENT_STATUS: 4
                   API_SEQ_NUM: 1
                 EMSX_SEQUENCE: 342481
                 EMSX_ROUTE_ID: 0
                  EMSX_FILL_ID: 0
                     EMSX_SIDE: {''}
                  EMSX_AMOUNT: 300
                  EMSX_FILLED: 0
                EMSX_AVG_PRICE: 0
                  EMSX_BROKER: {''}
                 EMSX_WORKING: 0
                  EMSX_TICKER: {'IBM US Equity'}
```

```
...
```

**Create the request structure.**

Create the request for the specific buy order for IBM® stock.

```
reqStruct.EMSX_TICKER = 'IBM';
reqStruct.EMSX_AMOUNT = int32(400);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';

% For limit orders, limit price can be set
% reqStruct.EMSX_LIMIT_PRICE = 150;
```

**Create the order.**

Create a new order.

```
rCreateOrder = b.createOrder(reqStruct)
```

```
rCreateOrder =

    EMSX_SEQUENCE: 344700
          MESSAGE: 'Order created'
```

Get the order status.

```
b.getOrderInfo(344700)
```

```
rOrderStatus1 =

               EMSX_TICKER: 'IBM'
             EMSX_EXCHANGE: 'US'
                 EMSX_SIDE: 'BUY'
             EMSX_POSITION: 'BUY'
             EMSX_PORT_MGR: 'CF'
               EMSX_TRADER: 'CF'
```

```
                EMSX_NOTES: ''
               EMSX_AMOUNT: 400
          EMSX_IDLE_AMOUNT: 0
              EMSX_WORKING: 200
               EMSX_FILLED: 200
            EMSX_TS_ORDNUM: 200
          EMSX_LIMIT_PRICE: 0
            EMSX_AVG_PRICE: 189.5900
                 EMSX_FLAG: 2
             EMSX_SUB_FLAG: 0
           EMSX_YELLOW_KEY: 'Equity'
          EMSX_BASKET_NAME: ''
    EMSX_ORDER_CREATE_DATE: '12/06/12'
    EMSX_ORDER_CREATE_TIME: '14:28:37'
           EMSX_ORDER_TYPE: 'MKT'
                  EMSX_TIF: 'DAY'
               EMSX_BROKER: 'BB'
          EMSX_TRADER_UUID: '1244972'
        EMSX_STEP_OUT_BROKER: ''
```

**Modify the order.**

Change the properties for an existing order and then route the order.

```
modStruct.EMSX_SEQUENCE = rCreateOrder.EMSX_SEQUENCE;
modStruct.EMSX_TICKER = 'IBM';
modStruct.EMSX_AMOUNT = int32(300);
rModifyOrder = b.modifyOrder(modStruct)

%Route order
% routeStruct.EMSX_AMOUNT = modStruct.EMSX_AMOUNT;
% routeStruct.EMSX_SEQUENCE = rModifyOrder.EMSX_SEQUENCE;
% routeStruct.EMSX_TICKER = reqStruct.EMSX_TICKER;
% routeStruct.EMSX_ORDER_TYPE = reqStruct.EMSX_ORDER_TYPE;
% routeStruct.EMSX_BROKER = reqStruct.EMSX_BROKER;
% routeStruct.EMSX_TIF = reqStruct.EMSX_TIF;
% routeStruct.EMSX_HAND_INSTRUCTION = reqStruct.EMSX_HAND_INSTRUCTION;
% routeStruct.EMSX_ODD_LOT = '-1';
% routeStruct.EMSX_CFD_FLAG = '-1';
% routeStruct.EMSX_RELEASE_TIME = '-1';
```

```
% rRouteOrder = b.routeOrder(routeStruct);

rModifyOrder =

    EMSX_SEQUENCE: 344700
          MESSAGE: 'Order Modified'
```

Get the modified order status.

```
rOrderStatus2 = b.getOrderInfo(344700)

rOrderStatus2 =

               EMSX_TICKER: 'IBM'
             EMSX_EXCHANGE: 'US'
                 EMSX_SIDE: 'BUY'
             EMSX_POSITION: 'BUY'
             EMSX_PORT_MGR: 'CF'
               EMSX_TRADER: 'CF'
                EMSX_NOTES: ''
               EMSX_AMOUNT: 300
          EMSX_IDLE_AMOUNT: 0
              EMSX_WORKING: 200
               EMSX_FILLED: 100
            EMSX_TS_ORDNUM: 200
          EMSX_LIMIT_PRICE: 0
            EMSX_AVG_PRICE: 189.5900
                 EMSX_FLAG: 2
             EMSX_SUB_FLAG: 0
           EMSX_YELLOW_KEY: 'Equity'
          EMSX_BASKET_NAME: ''
    EMSX_ORDER_CREATE_DATE: '12/06/12'
    EMSX_ORDER_CREATE_TIME: '14:28:37'
           EMSX_ORDER_TYPE: 'MKT'
                  EMSX_TIF: 'DAY'
               EMSX_BROKER: 'BB'
          EMSX_TRADER_UUID: '1244972'
      EMSX_STEP_OUT_BROKER: ''
```

**Delete the order (if necessary).**

The structure returned from the `createOrder` call can be used as the input to delete the order or you can create a new structure where the field `EMSX_SEQUENCE` contains the order number to be canceled.

```
delStruct.EMSX_SEQUENCE = rCreateOrder.EMSX_SEQUENCE;
rDeleteOrder = b.deleteOrder(delStruct)
```

```
rDeleteOrder =

     STATUS: 'O'
    MESSAGE: 'Order deleted'
```

**Close the connection.**

```
close(b)
processEvent(b)
```

```
SessionConnectionDown = {
    server = localhost/127.0.0.1:8194
}
```

## See Also

`createOrder` | `orders` | `modifyOrder` | `deleteOrder` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# Bloomberg EMSX Route Management

This example shows how to connect to a Bloomberg EMSX service, set up a route subscription, and create and manage a route.

**Connect to Bloomberg EMSX.**

```
b = emsx('//blp/emapisvc_beta');
processEvent(b)
```

```
SessionConnectionUp = {
    server = localhost/127.0.0.1:8194
}

SessionStarted = {
}

ServiceOpened = {
    serviceName = //blp/emapisvc_beta
}
```

**Set up the route subscription.**

```
rRouteStatus1 = b.routes({'EMSX_BROKER','EMSX_WORKING'})
```

```
rRouteStatus1 =

                      MSG_TYPE: {4x1 cell}
                  MSG_SUB_TYPE: {4x1 cell}
                  EVENT_STATUS: [4x1 int32]
                   API_SEQ_NUM: [4x1 int64]
                 EMSX_SEQUENCE: [4x1 int32]
                 EMSX_ROUTE_ID: [4x1 int32]
                  EMSX_FILL_ID: [4x1 int32]
                     EMSX_SIDE: {4x1 cell}
                  EMSX_AMOUNT: [4x1 int32]
                  EMSX_FILLED: [4x1 int32]
                EMSX_AVG_PRICE: [4x1 double]
                  EMSX_BROKER: {4x1 cell}
                 EMSX_WORKING: [4x1 int32]
                  EMSX_TICKER: {4x1 cell}
```

```
...
```

**Create the request structure.**

Create the request for a specific buy order for IBM stock.

```
reqStruct.EMSX_TICKER = 'IBM';
reqStruct.EMSX_AMOUNT = int32(3358);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';

% create and route order
rOrder = b.createOrderAndRoute(reqStruct)
```

```
rOrder =

    EMSX_SEQUENCE: 348930
    EMSX_ROUTE_ID: 1
          MESSAGE: 'Order created and routed'
```

Check the route status:

```
routeStruct.EMSX_SEQUENCE = rOrder.EMSX_SEQUENCE
routeStruct.EMSX_ROUTE_ID = rOrder.EMSX_ROUTE_ID

rRouteStatus2 = b.getRouteInfo(routeStruct)
```

```
rRouteStatus2 =

                  EMSX_AVG_PRICE: 189.5900
                      EMSX_YIELD: 0
         EMSX_ROUTE_CREATE_DATE: 20121206
         EMSX_ROUTE_CREATE_TIME: 142837
    EMSX_ROUTE_LAST_UPDATE_DATE: 20121206
    EMSX_ROUTE_LAST_UPDATE_TIME: 142838
                EMSX_SETTLE_DATE: 20121211
                     EMSX_AMOUNT: 400
```

```
                EMSX_FILLED: 200
     EMSX_IS_MANUAL_ROUTE: 0
                EMSX_BROKER: 'BB'
               EMSX_ACCOUNT: ''
             EMSX_STATUS_ID: 51088
                EMSX_STATUS: 'PtlFil'
EMSX_HAND_INSTRUCTION: 'ANY'
            EMSX_ORDER_TYPE: 'MKT'
                   EMSX_TIF: 'DAY'
                EMSX_LOC_ID: ''
            EMSX_LOC_BROKER: 'DAY'
            EMSX_STOP_PRICE: 0
         EMSX_BLOT_SEQ_NUM: 1
             EMSX_BLOT_DATE: 20121206
             EMSX_COMM_TYPE: 'DAY'
             EMSX_COMM_RATE: 0
EMSX_USER_COMM_AMOUNT: 0
             EMSX_LSTTR2ID0: 1.3548e+09
             EMSX_LSTTR2ID1: 284950536
            EMSX_LIMIT_PRICE: 0
```

**Modify the route.**

Modify the properties for the previously created route.

```
modStruct.EMSX_SEQUENCE = rOrder.EMSX_SEQUENCE;
modStruct.EMSX_ROUTE_ID = rOrder.EMSX_ROUTE_ID;
modStruct.EMSX_TICKER = 'IBM';
modStruct.EMSX_AMOUNT = int32(3000);
modStruct.EMSX_ORDER_TYPE = 'MKT';
modStruct.EMSX_TIF = 'DAY';
rModifyRoute = b.modifyRoute(modStruct);
```

Check the route status for the modified route.

```
rRouteStatus3 = b.getRouteInfo(routeStruct)
```

```
rRouteStatus3 =

             EMSX_AVG_PRICE: 189.7900
                 EMSX_YIELD: 0
```

```
     EMSX_ROUTE_CREATE_DATE: 20121206
     EMSX_ROUTE_CREATE_TIME: 142837
EMSX_ROUTE_LAST_UPDATE_DATE: 20121206
EMSX_ROUTE_LAST_UPDATE_TIME: 143251
           EMSX_SETTLE_DATE: 20121211
                EMSX_AMOUNT: 250
                EMSX_FILLED: 250
       EMSX_IS_MANUAL_ROUTE: 0
                EMSX_BROKER: 'BB'
               EMSX_ACCOUNT: ''
             EMSX_STATUS_ID: 199032
                EMSX_STATUS: 'Filled'
      EMSX_HAND_INSTRUCTION: 'ANY'
            EMSX_ORDER_TYPE: 'MKT'
                   EMSX_TIF: 'DAY'
                EMSX_LOC_ID: ''
            EMSX_LOC_BROKER: 'DAY'
            EMSX_STOP_PRICE: 0
          EMSX_BLOT_SEQ_NUM: 1
             EMSX_BLOT_DATE: 20121206
             EMSX_COMM_TYPE: 'DAY'
             EMSX_COMM_RATE: 0
      EMSX_USER_COMM_AMOUNT: 0
             EMSX_LSTTR2ID0: 1.3548e+09
             EMSX_LSTTR2ID1: 284950536
           EMSX_LIMIT_PRICE: 0
```

**Delete the route.**

The structure returned from the `createOrderAndRoute` call can be used as the input to delete the route or you can create a new structure where the field `EMSX_SEQUENCE` contains the order number to be canceled.

```
delStruct.EMSX_SEQUENCE = rOrder.EMSX_SEQUENCE;
delStruct.EMSX_ROUTE_ID = rOrder.EMSX_ROUTE_ID;
rDeleteRoute = b.deleteRoute(delStruct)
```

```
rDeleteRoute =

      STATUS: '0'
```

```
     MESSAGE: 'Route deleted'
```

**Close the connection.**

```
close(b)
processEvent(b)

SessionConnectionDown = {
    server = localhost/127.0.0.1:8194
}
```

## See Also

`createOrderAndRoute` | `modifyRoute` | `deleteRoute` | `routes` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# Bloomberg EMSX Order and Route Management

This example shows how to connect to a Bloomberg EMSX service, set up an order and route subscription, and create and manage an order and route.

**Connect to Bloomberg EMSX.**

```
b = emsx('//blp/emapisvc_beta');
processEvent(b)

SessionConnectionUp = {
    server = localhost/127.0.0.1:8194
}

SessionStarted = {
}

ServiceOpened = {
    serviceName = //blp/emapisvc_beta
}
```

**Set up the order and route subscription.**

```
o = b.orders({'EMSX_TICKER','EMSX_AMOUNT','EMSX_FILL'})
r = b.routes({'EMSX_BROKER','EMSX_WORKING'})

o =

                     MSG_TYPE: {4x1 cell}
                 MSG_SUB_TYPE: {4x1 cell}
                 EVENT_STATUS: [4x1 int32]
                  API_SEQ_NUM: [4x1 int64]
                EMSX_SEQUENCE: [4x1 int32]
                EMSX_ROUTE_ID: [4x1 int32]
                 EMSX_FILL_ID: [4x1 int32]
                    EMSX_SIDE: {4x1 cell}
                 EMSX_AMOUNT: [4x1 int32]
                 EMSX_FILLED: [4x1 int32]
              EMSX_AVG_PRICE: [4x1 double]
                 EMSX_BROKER: {4x1 cell}
                EMSX_WORKING: [4x1 int32]
```

```
                    EMSX_TICKER: {4x1 cell}
                  EMSX_EXCHANGE: {4x1 cell}

                  ...

r =

                        MSG_TYPE: {2x1 cell}
                    MSG_SUB_TYPE: {2x1 cell}
                    EVENT_STATUS: [2x1 int32]
                     API_SEQ_NUM: [2x1 int64]
                  EMSX_SEQUENCE: [2x1 int32]
                  EMSX_ROUTE_ID: [2x1 int32]
                   EMSX_FILL_ID: [2x1 int32]
                      EMSX_SIDE: {2x1 cell}
                     EMSX_AMOUNT: [2x1 int32]
                     EMSX_FILLED: [2x1 int32]
                 EMSX_AVG_PRICE: [2x1 double]
                     EMSX_BROKER: {2x1 cell}
                    EMSX_WORKING: [2x1 int32]
                     EMSX_TICKER: {2x1 cell}

                  ...
```

**Create the request structure.**

Create a request for a specific buy order for IBM stock.

```
reqStruct.EMSX_TICKER = 'IBM';
reqStruct.EMSX_AMOUNT = int32(400);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';

%For Limit orders, limit price can be set
%reqStruct.EMSX_LIMIT_PRICE = 150;
```

**Create the order and route.**

Create an order and route for execution.

```
rCreateOrderAndRoute = b.createOrderAndRoute(reqStruct)
```

```
rCreateOrderAndRoute =

    EMSX_SEQUENCE: 344705
    EMSX_ROUTE_ID: 1
          MESSAGE: 'Order created and routed'
```

Get the order status.

```
rOrderStatus1 = b.getOrderInfo(rCreateOrderAndRoute.EMSX_SEQUENCE)
```

```
rOrderStatus1 =

               EMSX_TICKER: 'IBM'
             EMSX_EXCHANGE: 'US'
                 EMSX_SIDE: 'BUY'
             EMSX_POSITION: 'BUY'
             EMSX_PORT_MGR: 'CG'
               EMSX_TRADER: 'CG'
                EMSX_NOTES: ''
               EMSX_AMOUNT: 400
          EMSX_IDLE_AMOUNT: 0
              EMSX_WORKING: 200
               EMSX_FILLED: 200
            EMSX_TS_ORDNUM: 200
          EMSX_LIMIT_PRICE: 0
            EMSX_AVG_PRICE: 189.5900
                 EMSX_FLAG: 2
             EMSX_SUB_FLAG: 0
           EMSX_YELLOW_KEY: 'Equity'
          EMSX_BASKET_NAME: ''
    EMSX_ORDER_CREATE_DATE: '12/06/12'
    EMSX_ORDER_CREATE_TIME: '14:28:37'
           EMSX_ORDER_TYPE: 'MKT'
                  EMSX_TIF: 'DAY'
               EMSX_BROKER: 'BB'
          EMSX_TRADER_UUID: '1244972'
```

Get the route status.

```
routeStat.EMSX_SEQUENCE = rCreateOrderAndRoute.EMSX_SEQUENCE
routeStat.EMSX_ROUTE_ID = rCreateOrderAndRoute.EMSX_ROUTE_ID

rRouteStatus1 = b.getRouteInfo(routeStat)
```

```
rRouteStatus1 =

                 EMSX_AVG_PRICE: 189.5900
                     EMSX_YIELD: 0
         EMSX_ROUTE_CREATE_DATE: 20121206
         EMSX_ROUTE_CREATE_TIME: 142837
    EMSX_ROUTE_LAST_UPDATE_DATE: 20121206
    EMSX_ROUTE_LAST_UPDATE_TIME: 142838
               EMSX_SETTLE_DATE: 20121211
                    EMSX_AMOUNT: 400
                    EMSX_FILLED: 200
           EMSX_IS_MANUAL_ROUTE: 0
                    EMSX_BROKER: 'BB'
                   EMSX_ACCOUNT: ''
                 EMSX_STATUS_ID: 51088
                    EMSX_STATUS: 'PtlFil'
          EMSX_HAND_INSTRUCTION: 'ANY'
                EMSX_ORDER_TYPE: 'MKT'
                        EMSX_TIF: 'DAY'
                    EMSX_LOC_ID: ''
                EMSX_LOC_BROKER: 'DAY'
                EMSX_STOP_PRICE: 0
              EMSX_BLOT_SEQ_NUM: 1
                 EMSX_BLOT_DATE: 20121206
                 EMSX_COMM_TYPE: 'DAY'
                 EMSX_COMM_RATE: 0
          EMSX_USER_COMM_AMOUNT: 0
                 EMSX_LSTTR2ID0: 1.3548e+09
                 EMSX_LSTTR2ID1: 284950536
                EMSX_LIMIT_PRICE: 0
```

**Modify the order on route.**

Modify the previously routed order.

```
modStruct.EMSX_SEQUENCE = rCreateOrderAndRoute.EMSX_SEQUENCE;
modStruct.EMSX_ROUTE_ID = rCreateOrderAndRoute.EMSX_ROUTE_ID;
modStruct.EMSX_TICKER = 'IBM';
modStruct.EMSX_AMOUNT = int32(250);
modStruct.EMSX_ORDER_TYPE = 'MKT';
modStruct.EMSX_TIF = 'DAY';
rModifyRoute = b.modifyRoute(modStruct)
```

```
rModifyRoute =

    EMSX_SEQUENCE: 0
    EMSX_ROUTE_ID: 0
          MESSAGE: 'Route modified'
```

**Delete order.**

The structure returned from the `createOrderAndRoute` call can be used as the input to delete the order or you can create a new structure where the field `EMSX_SEQUENCE` contains the order number to be canceled.

```
delStruct.EMSX_SEQUENCE = rCreateOrderAndRoute.EMSX_SEQUENCE;
delStruct.EMSX_ROUTE_ID = rCreateOrderAndRoute.EMSX_ROUTE_ID;
rDeleteOrder = b.deleteOrder(delStruct)
```

```
rDeleteOrder =

      STATUS: '0'
     MESSAGE: 'Order deleted'
```

**Close the connection.**

```
close(b)
processEvent(b)
```

```
SessionConnectionDown = {
    server = localhost/127.0.0.1:8194
}
```

## See Also

`createOrderAndRoute` | `orders` | `modifyOrder` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples


- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19


## Concepts


- "Workflow for Bloomberg EMSX" on page 3-2

5

# Functions — Alphabetical List

# emsx

## Purpose

Create Bloomberg EMSX connection

## Syntax

```
C = emsx(servicename)
```

## Description

`C = emsx(servicename)` creates a connection to the local Bloomberg EMSX communications server and uses the service `servicename`.

## Input Arguments

### servicename - Bloomberg EMSX service name

string

Bloomberg EMSX service name, specified using a test or production Bloomberg EMSX `servicename`.

**Data Types**

`char`

## Output Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, returned as an object structure.

## Examples

### Connect to the Test Bloomberg EMSX Service

Connect to test calls to the Bloomberg EMSX test service.

```
C = emsx('//blp/emapisvc_beta')
```

```
C 

  emsx with properties:

      Session: [1x1 com.bloomberglp.blpapi.Session]
      Service: [1x1 com.bloomberglp.blpapi.impl.aQ]
    Ipaddress: 'localhost'
         Port: 8194
```

### Connect to the Bloomberg EMSX Production Service

Connect to place “live” calls to the Bloomberg EMSX production service.

```
C = emsx('//bmp/emapisvc')

C = 

  emsx with properties:

      Session: [1x1 com.bloomberglp.blpapi.Session]
      Service: [1x1 com.bloomberglp.blpapi.impl.aQ]
    Ipaddress: 'localhost'
         Port: 8194
```

## See Also

`createOrder` | `createOrderAndRoute` | `close`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

# close

## Purpose

Close Bloomberg EMSX connection

## Syntax

```
close(C)
```

## Description

`close(C)` closes a connection to Bloomberg EMSX.

## Input Arguments

**`C` - Connection object for Bloomberg EMSX service**
object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

## Examples

### Close the Connection to the Bloomberg EMSX Service

Close connection `C`:

```
close(C)
```

## See Also

`emsx`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

## Purpose

Create Bloomberg EMSX order

## Syntax

```
R = createOrder(C,reqStruct)
R = createOrder(C,reqStruct,Name,Value)
```

## Description

`R = createOrder(C,reqStruct)` creates a Bloomberg EMSX order and returns the order sequence number and status message using the default event handler.

`R = createOrder(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Create a Bloomberg EMSX order using the optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
p.useDefaultEventHandler = false;
createOrder(C,reqStruct,p)
C.processEvent
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
createOrder(C,reqStruct,'useDefaultEventHandler',false)
```

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return for order status**
structure

Return for order status, returned as a structure.

## Examples

### Create Bloomberg EMSX Order Using Default Event Handler

Define the order request structure and create the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
r = createOrder(C,reqStruct)
```

```
 r =

 EMSX_SEQUENCE: 354646
 MESSAGE: 'Order created'
```

### Create Bloomberg EMSX Order Using Custom Event Handler

Define the order request structure and create the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

```
createOrder(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)

 MESSAGE: 'Order created' CreateOrder = {

     EMSX_SEQUENCE = 354651

     MESSAGE = Order created

     }
```

### Create Bloomberg EMSX Order Using `timeOut` Value

Define the order request structure and create the order specifying the `timeOut` value of 200 milliseconds.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
createOrder(C,reqStruct,'timeOut',200)

r =

 EMSX_SEQUENCE: 354646
```

## See Also

`createOrderAndRoute` | `orders` | `modifyOrder` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

## Purpose

Create and route Bloomberg EMSX order

## Syntax

```
R = createOrderAndRoute(C,reqStruct)
R = createOrderAndRoute(C,reqStruct,Name,Value)
```

## Description

`R = createOrderAndRoute(C,reqStruct)` creates and routes a Bloomberg EMSX order and returns the order sequence number, route ID, and status message using the default event handler.

`R = createOrderAndRoute(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Create and route a Bloomberg EMSX order with optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
p.useDefaultEventHandler = false;
createOrderAndRoute(C,reqStruct,p)
C.processEvent
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:** `reqStruct.EMSX_TICKER = 'XYZ';`
`reqStruct.EMSX_AMOUNT = int32(100);`
`reqStruct.EMSX_ORDER_TYPE = 'MKT';`
`reqStruct.EMSX_TIF = 'DAY';`
`reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';`
`reqStruct.EMSX_SIDE = 'BUY';`

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`createOrderAndRoute(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

`500` milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status for order event**
structure

Return status for the order event, returned as a structure.

## Examples

### Create and Route Bloomberg EMSX Order Using Default Event Handler

Define the order request structure and create and then route the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
r = createOrderAndRoute(C,reqStruct)
```

```
  r =

    EMSX_SEQUENCE: 335877
    EMSX_ROUTE_ID: 1
          MESSAGE: 'Order created and routed'
```

### Create and Route Bloomberg EMSX Order Using Custom Event Handler

Define the order request structure and create and then route the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
```

```
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
createOrderAndRoute(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
 CreateOrderAndRoute = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Order created and routed

     }
```

### Create and Route Bloomberg EMSX Order Using `timeOut` Value

Define the order request structure. Then create and route the order and assign a `timeOut` value of 200 milliseconds.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
createOrderAndRoute(C,reqStruct,'timeOut',200)
```

```
 r =

    EMSX_SEQUENCE: 335877
    EMSX_ROUTE_ID: 1
          MESSAGE: 'Order created and routed'
```

## See Also

`createOrder` | `createOrderAndRouteWithStrat` | `orders` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# createOrderAndRouteWithStrat

## Purpose

Create and route Bloomberg EMSX order with strategies

## Syntax

```
R = createOrderAndRoute(C,reqStruct,stratStruct)
R = createOrderAndRoute(C,reqStruct,stratStruct,Name,Value)
```

## Description

`R = createOrderAndRoute(C,reqStruct,stratStruct)` creates and routes a Bloomberg EMSX order with strategies and returns the order sequence number, route ID, and status message using the default event handler.

`R = createOrderAndRoute(C,reqStruct,stratStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Create and route a Bloomberg EMSX order with strategies using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
createOrderAndRouteWithStrat(C,reqStruct,stratStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:** `reqStruct.EMSX_TICKER = 'XYZ';`
`reqStruct.EMSX_AMOUNT = int32(100);`
`reqStruct.EMSX_ORDER_TYPE = 'MKT';`
`reqStruct.EMSX_TIF = 'DAY';`
`reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';`
`reqStruct.EMSX_SIDE = 'BUY';`

**Data Types**
`struct`

### stratStruct - Order strategies structure
structure

Order strategies structure, specified by the elements of the fields `EMSX_STRATEGY_FIELD_INDICATORS` and `EMSX_STRATEGY_FIELDS` in the `stratStruct`. In addition, the field elements of `stratStruct` must align with the fields for the strategy specified by `STRATSTRUCT.EMSX_STRATEGY_NAME`. For more information on strategy fields and ordering, see `getBrokerInfo`.

When using `stratStruct`, set `STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS` equal to `0` for each field so that the field data setting in `STRATSTRUCT.EMSX_FIELD_DATA` is used. Also set `STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS` equal to `1` to ignore the data in `STRATSTRUCT.EMSX_FIELD_DATA`.

**Example:** `stratStruct.EMSX_STRATEGY_NAME = 'SSP';`
`stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);`
`stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};`

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `r = createOrderAndRouteWithStrat(C,reqStruct,stratStruct,'useDefaultEventHanc`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

### R - Return status for order event

structure

Return status for the order event returned as a structure.

## Examples

### Create and Route Bloomberg EMSX Order with Strategies Using Default Event Handler

Define the order request structure and strategies structure and then create and route the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
r = createOrderAndRouteWithStrat(C,reqStruct,stratStruct)
```

```
 r =

        EMSX_SEQUENCE: 335877
        EMSX_ROUTE_ID: 1
              MESSAGE: 'Order created and routed'
```

### Create and Route Bloomberg EMSX Order with Strategies Using Custom Event Handler

Define the order request structure and strategies structure and then create and route the order.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
```

```
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
r = createOrderAndRouteWithStrat(C,reqStruct,stratStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
CreateOrderAndRouteWithStrat = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Order created and routed

     }
```

### Create and Route Bloomberg EMSX Order with Strategies Using `timeOut` Value

Define the order request structure and then create and route the order and assign a `timeOut` value of 200 milliseconds.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
createOrderAndRoute(C,reqStruct,stratStruct,'timeOut',200)
```

```
 r =

        EMSX_SEQUENCE: 335877
        EMSX_ROUTE_ID: 1
              MESSAGE: 'Order created and routed'
```

## See Also

`getBrokerInfo` | `createOrder` | `orders` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# deleteOrder

## Purpose

Delete Bloomberg EMSX order

## Syntax

```
R = deleteOrder(C,reqStruct)
R = deleteOrder(C,reqStruct,Name,Value)
```

## Description

`R = deleteOrder(C,reqStruct)` deletes a Bloomberg EMSX order and returns a status message using the default event handler.

`R = deleteOrder(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Delete a Bloomberg EMSX order using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
deleteOrder(C,reqStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure | integer for `EMSX_SEQUENCE` number

Order request structure, specified as a `reqStruct` or `EMSX_SEQUENCE` number.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
```

```
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`int32` | `struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
deleteOrder(C,int32(335877),'useDefaultEventHandler',false)
```

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status for requested event**
structure

Return status for the order event, returned as a structure.

## Examples

### Delete Bloomberg EMSX Order Using Default Event Handler

Define the `EMSX_SEQUENCE` for the order and then delete the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
r = deleteOrder(C,reqStruct)
```

```
 r =

      STATUS: 'O'
     MESSAGE: 'Order deleted'
```

### Delete Bloomberg EMSX Order Using Custom Event Handler

Define the `EMSX_SEQUENCE` for the order and then delete the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
deleteOrder(C,int32(335877),'useDefaultEventHandler',false)
processEvent(C)
```

```
DeleteOrder = {

        STATUS = O

        MESSAGE = Order deleted

    }
```

### Delete Bloomberg EMSX Order Using `timeOut` Value

Define the `EMSX_SEQUENCE` for the order and then delete the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
deleteOrder(C,int32(335877),'timeOut',200)
```

```
r =

     STATUS: 'O'
```

## See Also

`createOrderAndRoute` | `orders` | `createOrder` | `routes` | `modifyOrder`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

# deleteRoute

## Purpose

Delete Bloomberg EMSX route

## Syntax

```
R = deleteRoute(C,reqStruct)
R = deleteRoute(C,reqStruct,Name,Value)
```

## Description

`R = deleteRoute(C,reqStruct)` deletes a Bloomberg EMSX route and returns a status message using the default event handler.

`R = deleteRoute(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Delete a Bloomberg EMSX route using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
deleteRoute(C,reqStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure | integer for `EMSX_SEQUENCE` number

Order request structure, specified as a `reqStruct` or `EMSX_SEQUENCE` number.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
```

```
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`int32` | `struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
deleteRoute(C,reqStruct,'useDefaultEventHandler',false)
```

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status for requested event**
structure

Return status for the order event returned as a structure.

## Examples

### Delete Route for Bloomberg EMSX Order Using Default Event Handler

Define the `reqStruct` values for `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` and then delete the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_ROUTE_ID = int32(1)
r = deleteRoute(C,reqStruct)
```

```
 r =

      STATUS: '0'
     MESSAGE: 'Route deleted'
```

### Delete Route for Bloomberg EMSX Order Using Custom Event Handler

Define the `reqStruct` values for `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` and then delete the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_ROUTE_ID = int32(1)
deleteRoute(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
DeleteRoute = {

        STATUS = 0

        MESSAGE = Route deleted

    }
```

### Delete Route for Bloomberg EMSX Order Using `timeOut` Value

Define the `reqStruct` values for `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` and then delete the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_ROUTE_ID = int32(1)
deleteRoute(C,int32(335877),'timeOut',200)
```

```
 r =

      STATUS: '0'
     MESSAGE: 'Route deleted'
```

## See Also

`createOrderAndRoute` | `orders` | `createOrder` | `routes` | `modifyRoute`

## Related Examples


- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24


## Concepts


- "Workflow for Bloomberg EMSX" on page 3-2

# getAllFieldMetaData

## Purpose

Obtain Bloomberg EMSX field information

## Syntax

```
R = getAllFieldMetaData(C)
```

## Description

`R = getAllFieldMetaData(C)` returns the Bloomberg EMSX field information given the connection handle `C`.

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

## Output Arguments

### R - Return information for all fields

structure

Return information, returned as a structure for all fields supported by Bloomberg EMSX service. This information is used to create a request structure (`reqStruct`) for orders.

## Examples

### Request All Field Information for EMSX

Request all fields supported by Bloomberg EMSX service.

```
R = getAllFieldMetaData(C)
```

```
R =

        EMSX_FIELD_NAME: {113x1 cell}
         EMSX_DISP_NAME: {113x1 cell}
              EMSX_TYPE: {113x1 cell}
             EMSX_LEVEL: [113x1 double]
               EMSX_LEN: [113x1 double]
```

where

```
{r.EMSX_FIELD_NAME{1} r.EMSX_DISP_NAME{1} r.EMSX_TYPE{1} r.EMSX_LEVEL(1) r.EMSX_LEN(1)}
```

```
 'MSG_TYPE'     'Msg Type'     'String'     [0]     [1]
```

## See Also

`emsx`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# getBrokerInfo

## Purpose

Obtain Bloomberg EMSX broker and strategy information

## Syntax

```
R = getBrokerInfo(C,reqStruct)
```

## Description

`R = getBrokerInfo(C,reqStruct)` obtains Bloomberg EMSX broker and strategy information and returns a status message using the default event handler.

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:** `reqStruct.EMSX_TICKER = 'ABCD US Equity';`

**Data Types**

`struct`

## Output Arguments

### R - Return status for requested event

structure

Return status for the order event, returned as a structure.

## Examples

### Obtain Broker Information for Bloomberg EMSX

Define the `reqstruct` for one item and then request broker information.

```
reqStruct.EMSX_TICKER = 'ABCD US Equity';
r = getBrokerInfo(C,reqStruct)
```

```
r =
```

```
        EMSX_BROKERS: {2x1 cell}
```

Define the `reqstruct` for two items and then request broker information.

```
reqStruct.EMSX_TICKER = 'ABCD US Equity';
reqStruct.EMSX_BROKER = 'BMTB';
r = getBrokerInfo(b,reqStruct)
```

```
 r =

        EMSX_STRATEGIES: {16x1 cell}
```

Define the `reqstruct` for three items then request broker information.

```
reqStruct.EMSX_TICKER = 'ABCD US Equity';
reqStruct.EMSX_BROKER = 'BMTB';
reqStruct.EMSX_STRATEGY = 'SSP';
r = getBrokerInfo(b,reqStruct)
```

```
FieldName: {3x1 cell}
Disable: {3x1 cell}
StringValue: {3x1 cell}
```

## See Also

`getRouteInfo` | `getOrderInfo` | `createOrder` | `createOrderAndRoute` | `orders` | `modifyOrder` | `routes` | `deleteOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# getOrderInfo

## Purpose

Obtain Bloomberg EMSX order information

## Syntax

```
R = getOrderInfo(C,reqStruct)
R = getOrderInfo(C,reqStruct,Name,Value)
```

## Description

`R = getOrderInfo(C,reqStruct)` obtains Bloomberg EMSX order information and returns a status message using the default event handler.

`R = getOrderInfo(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Obtain Bloomberg EMSX order information using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
getOrderInfo(C,reqStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure | integer for `EMSX_SEQUENCE` number

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Note** `EMSX_SEQUENCE` must denote an existing order sequence number.

**Example:** `reqStruct.EMSX_SEQUENCE = int32(335877);`

**Data Types**
`int32` | `struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `r = getOrderInfo(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

### R - Return status for requested event

structure

Return status for the order event, returned as a structure.

## Examples

### Obtain Order Information for Bloomberg EMSX Using Default Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
r = getOrderInfo(C,reqStruct)
```

```
r =

              EMSX_TICKER: 'IBM'
            EMSX_EXCHANGE: 'US'
                EMSX_SIDE: 'BUY'
            EMSX_POSITION: 'BUY'
            EMSX_PORT_MGR: 'CF'
              EMSX_TRADER: 'CF'
               EMSX_NOTES: ''
              EMSX_AMOUNT: 400
         EMSX_IDLE_AMOUNT: 150
             EMSX_WORKING: 0
              EMSX_FILLED: 250
           EMSX_TS_ORDNUM: 250
         EMSX_LIMIT_PRICE: 0
           EMSX_AVG_PRICE: 189.7900
                EMSX_FLAG: 2
            EMSX_SUB_FLAG: 0
          EMSX_YELLOW_KEY: 'Equity'
```

```
          EMSX_BASKET_NAME: ''
    EMSX_ORDER_CREATE_DATE: '12/06/12'
    EMSX_ORDER_CREATE_TIME: '14:28:37'
           EMSX_ORDER_TYPE: 'MKT'
                   EMSX_TIF: 'DAY'
               EMSX_BROKER: 'BB'
          EMSX_TRADER_UUID: '1244972'
      EMSX_STEP_OUT_BROKER: ''
```

### Obtain Order Information for Bloomberg EMSX Using Custom Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` and must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
r = getOrderInfo(C,reqStruct,'useDefaultEventHandler',false)

processEvent(C)

OrderRouteFields = {

    MSG_TYPE = E

    EVENT_STATUS = 1

    API_SEQ_NUM = 8

    EMSX_SEQUENCE = 0

    EMSX_AMOUNT = 0

    EMSX_FILLED = 0

    EMSX_AVG_PRICE = 0.0

    EMSX_WORKING = 0
```

```
    EMSX_TIME_STAMP = 0
    ...
}

OrderInfo = {
    EMSX_TICKER = IBM
    EMSX_EXCHANGE = US
    EMSX_SIDE = BUY
    EMSX_POSITION = BUY
    EMSX_PORT_MGR = CG
    EMSX_TRADER = CG
    EMSX_NOTES =
    EMSX_AMOUNT = 400
    EMSX_IDLE_AMOUNT = 150
    EMSX_WORKING = 0
    EMSX_FILLED = 250
    EMSX_TS_ORDNUM = -381490
    EMSX_LIMIT_PRICE = 0.0
    EMSX_AVG_PRICE = 189.7899963378906
```

```
    EMSX_FLAG = 2

    EMSX_SUB_FLAG = 0

    EMSX_YELLOW_KEY = Equity

    EMSX_BASKET_NAME =

    EMSX_ORDER_CREATE_DATE = 12/06/12

    EMSX_ORDER_CREATE_TIME = 14:28:37

    EMSX_ORDER_TYPE = MKT

    EMSX_TIF = DAY

    EMSX_BROKER = BB

    EMSX_TRADER_UUID = 1244972

    EMSX_STEP_OUT_BROKER =

}
```

### Obtain Order Information for Bloomberg EMSX Using `timeOut` Value

Define the `reqstruct` and note that `EMSX_SEQUENCE` must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
r = getOrderInfo(C,reqStruct,'timeOut',200)
```

```
r =

               EMSX_TICKER: 'IBM'
             EMSX_EXCHANGE: 'US'
                 EMSX_SIDE: 'BUY'
```

```
              EMSX_POSITION: 'BUY'
              EMSX_PORT_MGR: 'CF'
                EMSX_TRADER: 'CF'
                 EMSX_NOTES: ''
                EMSX_AMOUNT: 400
          EMSX_IDLE_AMOUNT: 150
              EMSX_WORKING: 0
                EMSX_FILLED: 250
            EMSX_TS_ORDNUM: 250
          EMSX_LIMIT_PRICE: 0
            EMSX_AVG_PRICE: 189.7900
                 EMSX_FLAG: 2
              EMSX_SUB_FLAG: 0
           EMSX_YELLOW_KEY: 'Equity'
          EMSX_BASKET_NAME: ''
    EMSX_ORDER_CREATE_DATE: '12/06/12'
    EMSX_ORDER_CREATE_TIME: '14:28:37'
           EMSX_ORDER_TYPE: 'MKT'
                   EMSX_TIF: 'DAY'
                EMSX_BROKER: 'BB'
          EMSX_TRADER_UUID: '1244972'
      EMSX_STEP_OUT_BROKER: ''
```

## See Also

`getRouteInfo` | `getBrokerInfo` | `createOrder` | `createOrderAndRoute` | `orders` | `modifyOrder` | `routes` | `deleteOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

## Purpose

Obtain Bloomberg EMSX route information

## Syntax

```
R = getRouteInfo(C,reqStruct)
R = getRouteInfo(C,reqStruct,Name,Value)
```

## Description

`R = getRouteInfo(C,reqStruct)` obtains Bloomberg EMSX route information and returns a status message using the default event handler.

`R = getRouteInfo(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Obtain Bloomberg EMSX route information using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
getRouteInfo(C,reqStruct,p)
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure | integer for `EMSX_SEQUENCE` and `EMSX_ROUTE_ID`

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Note** `EMSX_SEQUENCE` must denote an existing order sequence number and `EMSX_ROUTE_ID` must denote an existing route ID.

**Example:** `reqStruct.EMSX_SEQUENCE = int32(335877);`
`reqStruct.EMSX_ROUTE_ID = int32(1);`

**Data Types**
`int32` | `struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `r = getRouteInfo(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

### R - Return status for requested event

structure

Return status for the order event, returned as a structure.

## Examples

### Obtain Route Information for Bloomberg EMSX Using Default Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_ROUTE_ID = int32(1);
r = getRouteInfo(C,reqStruct)
```

```
r =

                 EMSX_AVG_PRICE: 189.7900
                     EMSX_YIELD: 0
         EMSX_ROUTE_CREATE_DATE: 20121206
         EMSX_ROUTE_CREATE_TIME: 142837
    EMSX_ROUTE_LAST_UPDATE_DATE: 20121206
    EMSX_ROUTE_LAST_UPDATE_TIME: 143251
                EMSX_SETTLE_DATE: 20121211
                     EMSX_AMOUNT: 250
                     EMSX_FILLED: 250
           EMSX_IS_MANUAL_ROUTE: 0
                     EMSX_BROKER: 'BB'
                    EMSX_ACCOUNT: ''
                 EMSX_STATUS_ID: 199032
```

```
                EMSX_STATUS: 'Filled'
EMSX_HAND_INSTRUCTION: 'ANY'
      EMSX_ORDER_TYPE: 'MKT'
              EMSX_TIF: 'DAY'
                EMSX_LOC_ID: ''
      EMSX_LOC_BROKER: 'DAY'
      EMSX_STOP_PRICE: 0
    EMSX_BLOT_SEQ_NUM: 1
       EMSX_BLOT_DATE: 20121206
       EMSX_COMM_TYPE: 'DAY'
       EMSX_COMM_RATE: 0
EMSX_USER_COMM_AMOUNT: 0
       EMSX_LSTTR2ID0: 1.3548e+09
       EMSX_LSTTR2ID1: 284950536
     EMSX_LIMIT_PRICE: 0
```

### Obtain Route Information for Bloomberg EMSX Using Custom Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_ROUTE_ID = int32(1);
r = getRouteInfo(C,reqStruct,'useDefaultEventHandler',false)

processEvent(C)

RouteInfo = {

    EMSX_LIMIT_PRICE = 0.0

    EMSX_YIELD = 0.0

    EMSX_AVG_PRICE = 193.9600067138672

    EMSX_ROUTE_CREATE_DATE = 20121211
```

```
EMSX_ROUTE_CREATE_TIME = 101324

EMSX_ROUTE_LAST_UPDATE_DATE = 20121211

EMSX_ROUTE_LAST_UPDATE_TIME = 101325

EMSX_SETTLE_DATE = 20121214

EMSX_AMOUNT = 100

EMSX_FILLED = 50

EMSX_IS_MANUAL_ROUTE = 0

EMSX_BROKER = BB

EMSX_ACCOUNT =

EMSX_STATUS_ID = 51088

EMSX_STATUS = PtlFil

EMSX_HAND_INSTRUCTION = ANY

EMSX_ORDER_TYPE = MKT

EMSX_TIF = DAY

EMSX_LOC_ID =

EMSX_LOC_BROKER =

EMSX_STOP_PRICE = 0.0

EMSX_BLOT_SEQ_NUM = 2
```

```
    EMSX_BLOT_DATE = 20121211

    EMSX_COMM_TYPE =

    EMSX_COMM_RATE = 0.0

    EMSX_USER_COMM_AMOUNT = 0.0

    EMSX_LSTTR2ID0 = 1355238804

    EMSX_LSTTR2ID1 = 284950539

}
```

### Obtain Route Information for Bloomberg EMSX Using `timeOut` Value

Define the `reqstruct` and note that `EMSX_SEQUENCE` and `EMSX_ROUTE_ID` must denote an existing order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_ROUTE_ID = int32(1);
r = getRouteInfo(C,reqStruct,'timeOut',200)
```

```
r =

                 EMSX_AVG_PRICE: 189.7900
                     EMSX_YIELD: 0
         EMSX_ROUTE_CREATE_DATE: 20121206
         EMSX_ROUTE_CREATE_TIME: 142837
    EMSX_ROUTE_LAST_UPDATE_DATE: 20121206
    EMSX_ROUTE_LAST_UPDATE_TIME: 143251
               EMSX_SETTLE_DATE: 20121211
                    EMSX_AMOUNT: 250
                    EMSX_FILLED: 250
           EMSX_IS_MANUAL_ROUTE: 0
                    EMSX_BROKER: 'BB'
                   EMSX_ACCOUNT: ''
```

```
            EMSX_STATUS_ID: 199032
               EMSX_STATUS: 'Filled'
EMSX_HAND_INSTRUCTION: 'ANY'
      EMSX_ORDER_TYPE: 'MKT'
                 EMSX_TIF: 'DAY'
               EMSX_LOC_ID: ''
      EMSX_LOC_BROKER: 'DAY'
      EMSX_STOP_PRICE: 0
    EMSX_BLOT_SEQ_NUM: 1
            EMSX_BLOT_DATE: 20121206
            EMSX_COMM_TYPE: 'DAY'
            EMSX_COMM_RATE: 0
EMSX_USER_COMM_AMOUNT: 0
            EMSX_LSTTR2ID0: 1.3548e+09
            EMSX_LSTTR2ID1: 284950536
          EMSX_LIMIT_PRICE: 0
```

## See Also

`getOrderInfo` | `getBrokerInfo` | `createOrder` | `createOrderAndRoute` | `orders` | `modifyOrder` | `routes` | `deleteOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# modifyOrder

## Purpose

Modify Bloomberg EMSX order

## Syntax

```
R = modifyOrder(C,reqStruct)
R = modifyOrder(C,reqStruct,Name,Value)
```

## Description

`R = modifyOrder(C,reqStruct)` modifies a Bloomberg EMSX order and returns a status message using the default event handler.

`R = modifyOrder(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Modify a Bloomberg EMSX order using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
modifyOrder(C,reqStruct,'useDefaultEventHandler',false)
C.processEvent
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
```

```
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`modifyOrder(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status for requested event**
structure

Return status for the order event, returned as a structure.

## Examples

### Modify Order for Bloomberg EMSX Using Default Event Handler

Define the `reqStruct` and then modify the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(200);
r = modifyOrder(C,reqStruct)
```

```
r =

     EMSX_SEQUENCE: 3335877
           MESSAGE: 'Order Modified'
```

### Modify Order for Bloomberg EMSX Using Custom Event Handler

Define the `reqStruct` and then modify the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(200);
modifyOrder(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
  ModifyOrder = {

     EMSX_SEQUENCE = 335877

     MESSAGE = Order Modified

     }
```

### Modify Order for Bloomberg Using `timeOut` Value

Define the `reqStruct` and then modify the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_ROUTE_ID = int32(1)
modifyOrder(C,int32(335877),'timeOut',200)
```

```
r =

     EMSX_SEQUENCE: 3335877
           MESSAGE: 'Order Modified'
```

## See Also

`createOrderAndRoute` | `orders` | `createOrder` | `routes` | `deleteOrder`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

# modifyRoute

## Purpose

Modify Bloomberg EMSX route

## Syntax

```
R = modifyRoute(C,reqStruct)
R = modifyRoute(C,reqStruct,Name,Value)
```

## Description

`R = modifyRoute(C,reqStruct)` modifies a Bloomberg EMSX route and returns a status message using the default event handler.

`R = modifyRoute(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Modify a Bloomberg EMSX route using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
modifyRoute(C,reqStruct,p)
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
```

```
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`modifyRoute(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status**
structure

Return status for the order event, returned as a structure.

## Examples

### Modify Route for Bloomberg EMSX Order Using Default Event Handler

Define the `reqStruct` and then modify the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(200);
r = modifyRoute(C,reqStruct)
```

```
 r =

     EMSX_SEQUENCE: 3335877
     EMSX_ROUTE_ID: 1
           MESSAGE: 'Order Modified'
```

### Modify Route for Bloomberg EMSX Order Using Custom Event Handler

Define the `reqStruct` and then modify the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(200);
modifyRoute(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
ModifyRoute = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Route Modified
```

```
    }
```

### Modify Route for Bloomberg Using `timeOut` Value

Define the `reqStruct` and then modify the route.

```
reqStruct.EMSX_SEQUENCE = int32(335877)
reqStruct.EMSX_ROUTE_ID = int32(1)
modifyRoute(C,int32(335877),'timeOut',200)
```

```
 r =

    EMSX_SEQUENCE: 3335877
    EMSX_ROUTE_ID: 1
          MESSAGE: 'Order Modified'
```

## See Also

`createOrderAndRoute` | `orders` | `createOrder` | `routes` | `deleteOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# modifyRouteWithStrat

## Purpose

Modify route with strategies for Bloomberg EMSX

## Syntax

```
R = modifyRouteWithStrat(C,reqStruct,stratStruct)
R =
modifyRouteWithStrat(C,reqStruct,stratStruct,Name,Value)
```

## Description

`R = modifyRouteWithStrat(C,reqStruct,stratStruct)` modifies a Bloomberg EMSX route with strategies and returns the order sequence number, route ID, and status message using the default event handler.

`R = modifyRouteWithStrat(C,reqStruct,stratStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Modify a Bloomberg EMSX route with strategies using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
modifyRouteWithStrat(C,reqStruct,stratStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:** `reqStruct.EMSX_TICKER = 'XYZ';`
`reqStruct.EMSX_AMOUNT = int32(100);`
`reqStruct.EMSX_ORDER_TYPE = 'MKT';`
`reqStruct.EMSX_TIF = 'DAY';`
`reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';`
`reqStruct.EMSX_SIDE = 'BUY';`

**Data Types**
`struct`

### stratStruct - Order strategies structure
structure

Order strategies structure specified by the elements of the fields `EMSX_STRATEGY_FIELD_INDICATORS` and `EMSX_STRATEGY_FIELDS` in the `stratStruct`. In addition, the field elements of `stratStruct` must align with the fields for the strategy specified by `STRATSTRUCT.EMSX_STRATEGY_NAME`. For more information on strategy fields and ordering, see `getBrokerInfo`.

When using `stratStruct`, set `STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS` equal to `0` for each field so that the field data setting in `STRATSTRUCT.EMSX_FIELD_DATA` is used. Also set `STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS` equal to `1` to ignore the data in `STRATSTRUCT.EMSX_FIELD_DATA`.

**Example:** `stratStruct.EMSX_STRATEGY_NAME = 'SSP';`
`stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);`
`stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};`

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `r = modifyRouteWithStrat(C,reqStruct,stratStruct,'useDefaultEventHandler',fal`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

`500` milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

### R - Return status for order event

structure

Return status for the order event, returned as a structure.

## Examples

### Modify Bloomberg EMSX Route with Strategies Using Default Event Handler

Define the order request structure and strategies structure and then modify the route.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
r = modifyRouteWithStrat(C,reqStruct,stratStruct)
```

```
 r =

        EMSX_SEQUENCE: 335877
        EMSX_ROUTE_ID: 1
              MESSAGE: 'Order Modified'
```

### Modify Bloomberg EMSX Route with Strategies Using Custom Event Handler

Define the order request structure and strategies structure and then modify the route.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
```

```
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
r = modifyRouteWithStrat(C,reqStruct,stratStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
ModifyRouteWithStrat = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Route modified

     }
```

### Modify Bloomberg EMSX Route with Strategies Using `timeOut` Value

Define the order request structure and modify route and assign a `timeOut` value of 200 milliseconds.

```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
modifyRouteWithStrat(C,reqStruct,stratStruct,'timeOut',200)
```

```
 r =

        EMSX_SEQUENCE: 335877
        EMSX_ROUTE_ID: 1
              MESSAGE: 'Order Modified'
```

## See Also

`getBrokerInfo` | `createOrderAndRouteWithStrat` | `createOrder` | `orders` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# orders

## Purpose

Obtain Bloomberg order subscription

## Syntax

```
R = orders(C,fields)
R = orders(C,fields,Name,Value)
```

## Description

`R = orders(C,fields)` subscribes to Bloomberg EMSX fields and returns information about outstanding orders using the default event handler.

`R = orders(C,fields,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Subscribe to Bloomberg EMSX fields and return information about outstanding orders using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
orders(C,{'EMSX_BROKER','EMSX_AMOUNT','EMSX_FILLED'},p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### fields - EMSX field information

cell array

EMSX field information, specified using a cell array. Use `getAllFieldMetaData` to view available field information for the Bloomberg EMSX service.

**Example:** `'EMSX_TICKER'`
`'EMSX_AMOUNT'`

'EMSX_ORDER_TYPE'

**Data Types**
cell

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`orders(C,{'EMSX_BROKER','EMSX_AMOUNT','EMSX_FILLED'},'useDefaultEventH`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
logical

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
char

## Output Arguments

**R - Return status**
structure

Return status for order subscription information for existing orders, returned as a structure.

## Examples

### Request Order Subscription for Bloomberg EMSX Orders Using Default Event Handler

Request order subscription for existing EMSX orders.

```
orders(C,{'EMSX_BROKER','EMSX_AMOUNT','EMSX_FILLED'})
```

```
orders =

                      MSG_TYPE: {7x1 cell}
                  MSG_SUB_TYPE: {7x1 cell}
                  EVENT_STATUS: [7x1 int32]
                   API_SEQ_NUM: [7x1 int64]
                 EMSX_SEQUENCE: [7x1 int32]
                 EMSX_ROUTE_ID: [7x1 int32]
                  EMSX_FILL_ID: [7x1 int32]
                     EMSX_SIDE: {7x1 cell}
                  EMSX_AMOUNT: [7x1 int32]
                  EMSX_FILLED: [7x1 int32]
                EMSX_AVG_PRICE: [7x1 double]
                  EMSX_BROKER: {7x1 cell}
                  EMSX_WORKING: [7x1 int32]
                  EMSX_TICKER: {7x1 cell}
                 EMSX_EXCHANGE: {7x1 cell}
        EMSX_ROUTE_CREATE_TIME: {7x1 cell}
                      EMSX_TIF: {7x1 cell}
   EMSX_ROUTE_LAST_UPDATE_TIME: {7x1 cell}

                     ....
```

### Request Order Subscription for Bloomberg EMSX Orders Using Custom Event Handler

Use the custom event handler.

```
orders(C,{'EMSX_BROKER','EMSX_AMOUNT','EMSX_FILLED'},'useDefaultEventHandler',false)
processEvent(C)
```

```
OrderRouteFields = {

    MSG_TYPE = E

    EVENT_STATUS = 1

    API_SEQ_NUM = 2

    EMSX_SEQUENCE = 0

    EMSX_AMOUNT = 0

    EMSX_FILLED = 0

    EMSX_AVG_PRICE = 0.0

    EMSX_WORKING = 0

    EMSX_TIME_STAMP = 0

    EMSX_ROUTE_PRICE = 0.0

    EMSX_LIMIT_PRICE = 0.0

    ...
```

### Request Order Subscription for Bloomberg EMSX Orders Using `timeOut` Value

Use the `timeOut` value.

```
orders(C,{'EMSX_BROKER','EMSX_AMOUNT','EMSX_FILLED'},'timeOut',200)
```

```
 orders =

                      MSG_TYPE: {7x1 cell}
                  MSG_SUB_TYPE: {7x1 cell}
                  EVENT_STATUS: [7x1 int32]
                   API_SEQ_NUM: [7x1 int64]
                 EMSX_SEQUENCE: [7x1 int32]
                 EMSX_ROUTE_ID: [7x1 int32]
                  EMSX_FILL_ID: [7x1 int32]
                     EMSX_SIDE: {7x1 cell}
                   EMSX_AMOUNT: [7x1 int32]
                   EMSX_FILLED: [7x1 int32]
                EMSX_AVG_PRICE: [7x1 double]
                   EMSX_BROKER: {7x1 cell}
                  EMSX_WORKING: [7x1 int32]
                   EMSX_TICKER: {7x1 cell}
                 EMSX_EXCHANGE: {7x1 cell}
         EMSX_ROUTE_CREATE_TIME: {7x1 cell}
                      EMSX_TIF: {7x1 cell}
    EMSX_ROUTE_LAST_UPDATE_TIME: {7x1 cell}

                                 ...
```

## See Also

`emsx` | `createOrder`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

## Purpose

Bloomberg EMSX example order blotter

## Syntax

```
[T,Subs] = emsxOrderBlotter(C)
```

## Description

`[T,Subs] = emsxOrderBlotter(C)` displays a trader's order information. `C` is the Bloomberg EMSX connection object, `T` is the timer object associated with the event handler, and `Subs` is the Bloomberg order subscription.

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

## Output Arguments

### `T` - Timer for event handler

string

Timer for the event handler, returned as a string.

### `Subs` - Bloomberg EMSX order subscription

structure

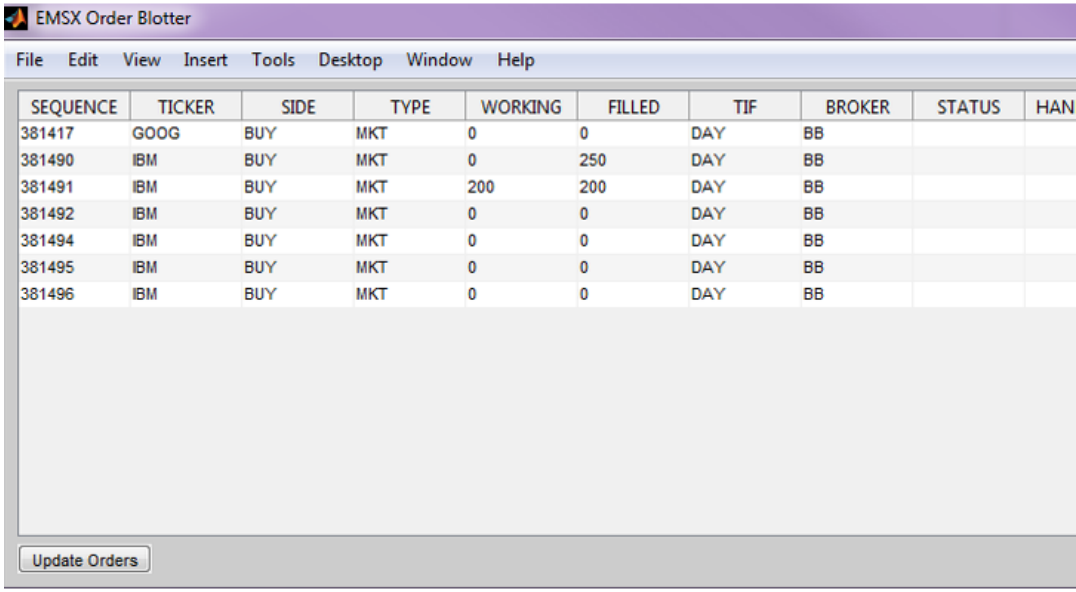
Bloomberg EMSX order subscription, returned as an object.

## Examples

### Display Order in Order Blotter Interface

Start the EMSX order blotter interface for connection object `C`.

```
emsxOrderBlotter(C)
```




| SEQUENCE | TICKER | SIDE | TYPE | WORKING | FILLED | TIF | BROKER | STATUS | HAN |
|---|---|---|---|---|---|---|---|---|---|
| 381417 | GOOG | BUY | MKT | 0 | 0 | DAY | BB | | |
| 381490 | IBM | BUY | MKT | 0 | 250 | DAY | BB | | |
| 381491 | IBM | BUY | MKT | 200 | 200 | DAY | BB | | |
| 381492 | IBM | BUY | MKT | 0 | 0 | DAY | BB | | |
| 381494 | IBM | BUY | MKT | 0 | 0 | DAY | BB | | |
| 381495 | IBM | BUY | MKT | 0 | 0 | DAY | BB | | |
| 381496 | IBM | BUY | MKT | 0 | 0 | DAY | BB | | |

The order blotter interface shows the current order information for a trader.

Define a `reqSruct` and then create a Bloomberg order.

```
reqStruct.EMSX_AMOUNT = int32(330);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
reqStruct.EMSX_TICKER = 'XYZ';
b.createOrderAndRoute(reqStruct,'useDefaultEventHandler',false)
```

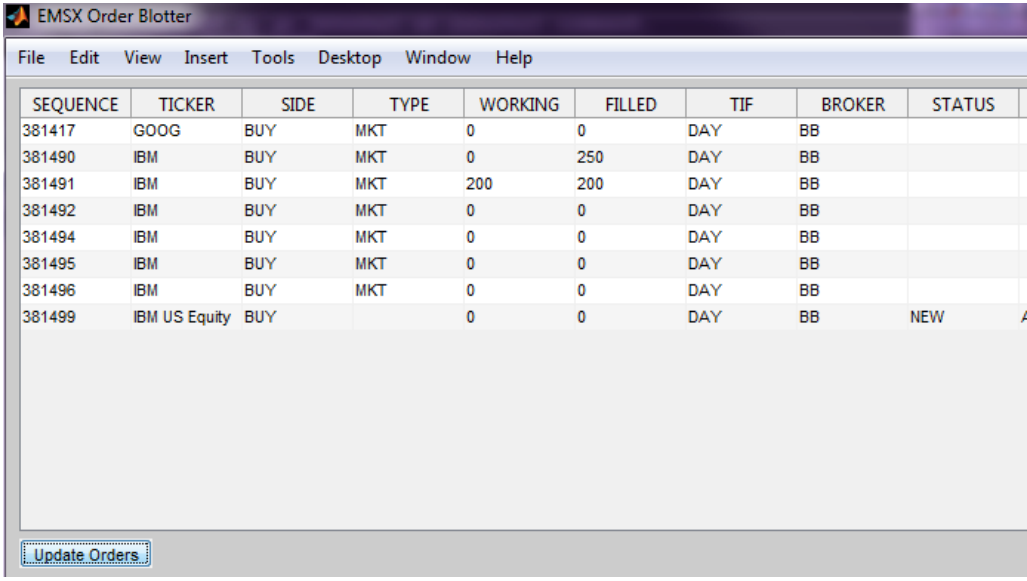
```
 CreateOrderAndRoute = {

     EMSX_SEQUENCE = 381499
```

```
EMSX_ROUTE_ID = 1

MESSAGE = Order created and routed

}
```




| SEQUENCE | TICKER | SIDE | TYPE | WORKING | FILLED | TIF | BROKER | STATUS |
|---|---|---|---|---|---|---|---|---|
| 381417 | GOOG | BUY | MKT | 0 | 0 | DAY | BB | |
| 381490 | IBM | BUY | MKT | 0 | 250 | DAY | BB | |
| 381491 | IBM | BUY | MKT | 200 | 200 | DAY | BB | |
| 381492 | IBM | BUY | MKT | 0 | 0 | DAY | BB | |
| 381494 | IBM | BUY | MKT | 0 | 0 | DAY | BB | |
| 381495 | IBM | BUY | MKT | 0 | 0 | DAY | BB | |
| 381496 | IBM | BUY | MKT | 0 | 0 | DAY | BB | |
| 381499 | IBM US Equity | BUY | | 0 | 0 | DAY | BB | NEW |

This updates the order blotter interface with information on the created and routed order (EMSX_SEQUENCE `381499`) using the event handler function `processEventToBlotter`. As orders are created and managed, the blotter is updated.

## See Also

`emsx` | `createOrder`

## Related Examples

- “Bloomberg EMSX Order Management” on page 4-14
- “Bloomberg EMSX Route Management” on page 4-19
- “Bloomberg EMSX Order and Route Management” on page 4-24

## Concepts

- “Workflow for Bloomberg EMSX” on page 3-2

# processEvent

## Purpose

Sample Bloomberg EMSX event handler

## Syntax

```
processEvent(C)
```

## Description

`processEvent(C)` processes the EMSX event queue associated with Bloomberg EMSX connection handle, `C`.

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

## Examples

### Continually Process the Bloomberg EMSX Event Queue

Use the following command to continually process the EMSX event queue.

```
T = timer('TimerFcn',{@b.processEvent},'Period',1,'ExecutionMode','fixedRate')
```

## See Also

`createOrderAndRoute` | `orders` | `modifyOrder` | `routes` | `deleteOrder` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

## Purpose

Route Bloomberg EMSX order

## Syntax

```
R = routeOrder(C,reqStruct)
R = routeOrder(C,reqStruct,Name,Value)
```

## Description

`R = routeOrder(C,reqStruct)` routes a Bloomberg EMSX order and returns a status message using the default event handler.

`R = routeOrder(C,reqStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Route a Bloomberg EMSX order using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
routeOrder(C,reqStruct,p)
```

---

## Input Arguments

### `C` - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### `reqStruct` - Order request structure

structure

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Example:**
```
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_TIF = 'DAY';
```

```
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
reqStruct.EMSX_SIDE = 'BUY';
```

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`routeOrder(C,reqStruct,'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

**R - Return status for requested event**
structure

Return status for the order event, returned as a structure.

## Examples

### Route Order for Bloomberg EMSX Using Default Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` must denote an existing order sequence number.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
r = routeOrder(C,reqStruct)
```

```
 r =

     EMSX_SEQUENCE: 335877
     EMSX_ROUTE_ID: 1
           MESSAGE: 'Order Routed'
```

### Route Order for Bloomberg EMSX Using Custom Event Handler

Define the `reqstruct` and note that `EMSX_SEQUENCE` must denote an existing order sequence number.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
```

```
routeOrder(C,reqStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
Route = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Order Routed

     }
```

### Route Order for Bloomberg EMSX Using `timeOut` Value

Define the `reqstruct` and note that `EMSX_SEQUENCE` must denote an existing order sequence number.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
routeOrder(C,reqStruct,'timeOut',200)
```

```
r =

     EMSX_SEQUENCE: 335877
     EMSX_ROUTE_ID: 1
           MESSAGE: 'Order Routed'
```

## See Also

`createOrder` | `createOrderAndRoute` | `orders` | `modifyOrder` | `routes` | `deleteOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

# routeOrderWithStrat

## Purpose

Route Bloomberg EMSX order with strategies

## Syntax

```
R = routeOrderWithStrat(C,reqStruct,stratStruct)
R = routeOrderWithStrat(C,reqStruct,stratStruct,Name,Value)
```

## Description

`R = routeOrderWithStrat(C,reqStruct,stratStruct)` routes a Bloomberg EMSX order with strategies and returns the order sequence number, route ID, and status message using the default event handler.

`R = routeOrderWithStrat(C,reqStruct,stratStruct,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Route a Bloomberg EMSX order with strategies using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
routeOrderWithStrat(C,reqStruct,stratStruct,p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### reqStruct - Order request structure

structure | integer for `EMSX_SEQUENCE` number

Order request structure, specified using EMSX field properties. Use `getAllFieldMetaData` to view all available field property options for `reqStruct`.

**Note** EMSX_SEQUENCE must denote an existing order sequence number.

**Example:**
```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
```

**Data Types**
int32 | struct

### stratStruct - Order strategies structure
structure

Order strategies structure specified by the elements of the fields EMSX_STRATEGY_FIELD_INDICATORS and EMSX_STRATEGY_FIELDS in the stratStruct. In addition, the field elements of stratStruct must align with the fields for the strategy specified by STRATSTRUCT.EMSX_STRATEGY_NAME. For more information on strategy fields and ordering, see getBrokerInfo.

When using stratStruct, set STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS equal to 0 for each field so that the field data setting in STRATSTRUCT.EMSX_FIELD_DATA is used. Also set STRATSTRUCT.EMSX_STRATEGY_FIELD_INDICATORS equal to 1 to ignore the data in STRATSTRUCT.EMSX_FIELD_DATA.

**Example:**
```
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0
0]);
stratStruct.EMSX_STRATEGY_FIELDS =
{'09:30:00','14:30:00',50};
```

**Data Types**
struct

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
`routeOrderWithStrat(C,reqStruct,stratStruct,'useDefaultEventHandler',fals`

#### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
`logical`

#### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
`char`

## Output Arguments

#### R - Return status for order event

structure

Return status for the order event, returned as a structure.

## Examples

### Route Bloomberg EMSX Order with Strategies Using Default Event Handler

Define the order request structure and strategies structure and then route the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
 r = routeOrderWithStrat(C,reqStruct,stratStruct)
```

```
 r =

     EMSX_SEQUENCE: 335877
     EMSX_ROUTE_ID: 1
           MESSAGE: 'Order Routed'
```

### Route Bloomberg EMSX Order with Strategies Using Custom Event Handler

Define the order request structure and strategies structure and then route the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
```

```
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
routeOrderWithStrat(C,reqStruct,stratStruct,'useDefaultEventHandler',false)
processEvent(C)
```

```
 Route = {

     EMSX_SEQUENCE = 335877

     EMSX_ROUTE_ID = 1

     MESSAGE = Order Routed

     }
```

### Route Bloomberg EMSX Order with Strategies Using `timeOut` Value

Define the order request structure and strategies structure and then route the order.

```
reqStruct.EMSX_SEQUENCE = int32(335877);
reqStruct.EMSX_TICKER = 'XYZ';
reqStruct.EMSX_AMOUNT = int32(100);
reqStruct.EMSX_ORDER_TYPE = 'MKT';
reqStruct.EMSX_BROKER = 'BB';
reqStruct.EMSX_TIF = 'DAY';
reqStruct.EMSX_HAND_INSTRUCTION = 'ANY';
stratStruct.EMSX_STRATEGY_NAME = 'SSP';
stratStruct.EMSX_STRATEGY_FIELD_INDICATORS = int32([0 0 0]);
stratStruct.EMSX_STRATEGY_FIELDS = {'09:30:00','14:30:00',50};
routeOrderWithStrat(C,reqStruct,stratStruct,'timeOut',200)
```

```
 r =

     EMSX_SEQUENCE: 335877
     EMSX_ROUTE_ID: 1
           MESSAGE: 'Order Routed'
```

## See Also

`createOrderAndRouteWithStrat` | `getRouteInfo` | `createOrder` | `orders` | `deleteOrder` | `routes` | `routeOrder`

## Related Examples


- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24


## Concepts


- "Workflow for Bloomberg EMSX" on page 3-2

## Purpose

Obtain Bloomberg EMSX route subscription

## Syntax

```
R = routes(C,fields)
R = routes(C,fields,Name,Value)
```

## Description

`R = routes(C,fields)` subscribes to Bloomberg EMSX fields and returns information about existing routes using the default event handler.

`R = routes(C,fields,Name,Value)` uses additional options specified by one or more `Name`, `Value` pair arguments. Subscribe to Bloomberg EMSX fields and return information about existing routes using optional name-value arguments to specify a custom event handler or timeout value for the event handler.

---

**Note** Optional name-value pair arguments can be input as a single input structure containing some or all of the property fields, for example:

```
p.timeOut = 1000;
routes(C,{'EMSX_BROKER','EMSX_WORKING'},p)
```

---

## Input Arguments

### C - Connection object for Bloomberg EMSX service

object structure

Connection object for Bloomberg EMSX service, specified using `emsx`.

### fields - EMSX field information

cell array

EMSX field information, specified using a cell array. Use `getAllFieldMetaData` to view available field information for the Bloomberg EMSX service.

**Example:** `'EMSX_TICKER'`
`'EMSX_AMOUNT'`

'EMSX_ORDER_TYPE'

**Data Types**
cell

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:** `routes(C, {'EMSX_BROKER','EMSX_WORKING'},'useDefaultEventHandler',false)`

### 'useDefaultEventHandler' - Flag for event handler preference

`true` (default) | logical with value `true` or `false`

Flag for event handler preference, indicating whether to use the default or custom event handler to process order events, as specified by the string `true` or `false`. When this flag is set to the default, `true`, the default event handler is used. If a custom event handler is used, this flag must be set to `false`.

**Example:** `'useDefaultEventHandler',false`

**Data Types**
logical

### 'timeOut' - Connection timeout value for event handler for Bloomberg EMSX service

500 milliseconds (default) | nonnegative integer

Connection timeout value, specified as a nonnegative integer in units of milliseconds.

**Example:** `'timeOut',200`

**Data Types**
char

## Output Arguments

**R - Return status for requested event**
structure

Return status for the route subscription for existing routes, returned as a structure.

## Examples

### Route Subscription for Bloomberg EMSX Using Default Event Handler

Return the status for route subscription for existing routes.

```
routes(C, {'EMSX_BROKER','EMSX_WORKING'})
```

```
routes =

                      MSG_TYPE: {3x1 cell}
                  MSG_SUB_TYPE: {3x1 cell}
                  EVENT_STATUS: [3x1 int32]
                   API_SEQ_NUM: [3x1 int64]
                 EMSX_SEQUENCE: [3x1 int32]
                 EMSX_ROUTE_ID: [3x1 int32]
                  EMSX_FILL_ID: [3x1 int32]
                     EMSX_SIDE: {3x1 cell}
                   EMSX_AMOUNT: [3x1 int32]
                   EMSX_FILLED: [3x1 int32]
                EMSX_AVG_PRICE: [3x1 double]
                   EMSX_BROKER: {3x1 cell}
                  EMSX_WORKING: [3x1 int32]
                   EMSX_TICKER: {3x1 cell}
                 EMSX_EXCHANGE: {3x1 cell}

                        ...
```

### Route Subscription for Bloomberg EMSX Using Custom Event Handler

Return the status for route subscription for existing routes using a custom event handler.

```
routes(C, {'EMSX_BROKER','EMSX_WORKING'},'useDefaultEventHandler',false)
processEvent(C)
```

```
OrderRouteFields = {

    MSG_TYPE = E

    MSG_SUB_TYPE = R

    EVENT_STATUS = 4

    API_SEQ_NUM = 1

    EMSX_SEQUENCE = 381490

    EMSX_ROUTE_ID = 1

    EMSX_FILL_ID = 0

   ...
```

### Route Subscription for Bloomberg EMSX Using `timeOut` Value

Return the status for route subscription for existing routes using a `timeout` value.

```
routes(C, {'EMSX_BROKER','EMSX_WORKING'},'timeOut',200)
```

```
routes =

                     MSG_TYPE: {3x1 cell}
                 MSG_SUB_TYPE: {3x1 cell}
                 EVENT_STATUS: [3x1 int32]
                  API_SEQ_NUM: [3x1 int64]
                EMSX_SEQUENCE: [3x1 int32]
                EMSX_ROUTE_ID: [3x1 int32]
                 EMSX_FILL_ID: [3x1 int32]
                    EMSX_SIDE: {3x1 cell}
```

```
                EMSX_AMOUNT: [3x1 int32]
                EMSX_FILLED: [3x1 int32]
             EMSX_AVG_PRICE: [3x1 double]
                EMSX_BROKER: {3x1 cell}
               EMSX_WORKING: [3x1 int32]
                EMSX_TICKER: {3x1 cell}
              EMSX_EXCHANGE: {3x1 cell}

                        ...
```

## See Also

`emsx` | `createOrderAndRoute` | `deleteRoute` | `modifyRoute` | `routeOrder`

## Related Examples

- "Bloomberg EMSX Order Management" on page 4-14
- "Bloomberg EMSX Route Management" on page 4-19
- "Bloomberg EMSX Order and Route Management" on page 4-24

## Concepts

- "Workflow for Bloomberg EMSX" on page 3-2

## Purpose

Create X_TRADER connection

## Syntax

```
X = xtrdr
```

## Description

`X = xtrdr` starts X_TRADER or connects to an existing X_TRADER session.

## Output Arguments

**X - Connection object**
object structure

Connection object for X_TRADER session.

## Limitations

- You should only create one X_TRADER connection per MATLAB session. To create a new X_TRADER connection, start a new MATLAB session.

## Examples

### Create a Connection to X_TRADER

```
X = xtrdr
```

```
x =

  xtrdr with properties:

           Gate: [1x1 COM.Xtapi_TTGate_1]
    InstrNotify: []
     Instrument: []
       OrderSet: []
```

## See Also

`close`

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

# close

**Purpose** Close X_TRADER connection

**Syntax** `close(X)`

**Description** `close(X)` closes the X_TRADER connection `X`.

**Input Arguments**

**`X` - Connection object**
object structure

Connection object for an X_TRADER session.

**Examples**

**Close X_TRADER Connection**

```
close(X)
```

**See Also** `xtrdr`

**Related Examples**

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

**Concepts**

- "Workflows for Trading Technologies X_TRADER" on page 3-4

## Purpose

Create instrument 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 - Connection object

object structure

Connection object, specified using `xtrdr`.

### S - `xtrdr` input structure

structure

`xtrdr` input structure, specified using 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*.

**Example:**
```
S = [];
S.Exchange = 'Eurex';
S.Product = 'OGBM';
S.ProdType = 'Option';
S.Contract = 'Jan12 P12300';
S.Alias = 'TestInstrument3';
```

**Data Types**

`struct`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**

```
createInstrument(X,'Exchange','Eurex','Product','OGBM','ProdType','Optior
P12300','Alias','TestInstrument3')
```

#### 'Property1' - Valid X_TRADER API options

string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

---

**Note** When using the `'alias'` property, ensure that every `'alias'` name is unique across all X_TRADER instruments.

---

**Data Types**

`char`

#### 'Property2' - Valid X_TRADER API options

string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Data Types**

`char`

## Examples

### Create an X_TRADER 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 X_TRADER 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` | `createNotifier` | `createOrderProfile` | `createOrderSet`

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

## Purpose

Create 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 - Connection object

object structure

Connection object, specified using `xtrdr`.

### S - `xtrdr` input structure with fields

structure

`xtrdr` input structure, specified 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*.

**Example:**
```
S = [];
S.Exchange = 'Eurex';
S.Product = 'OGBM';
S.ProdType = 'Option';
S.Contract = 'Jan12 P12300';
S.Alias = 'TestInstrument3';
```

**Data Types**

`struct`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
createNotifier(X,'Instrument',[],'UpdateFilter','','EnablePriceUpdates',-
```

#### 'Property1' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createNotifier(X,'Instrument',[],'UpdateFilter','','EnablePriceUpdates',-
```

**Data Types**
`char`

#### 'Property2' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createNotifier(X,'Instrument',[],'UpdateFilter','','EnablePriceUpdates',-
```

**Data Types**
`char`

## Examples

### Create an X_TRADER 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 X_TRADER 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` | `createInstrument` | `createOrderProfile` | `createOrderSet`

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

## Purpose

Create order profile 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

object structure

`xtrdr` connection object, specified using `xtrdr`.

### S - `xtrdr` input structure with fields

structure

`xtrdr` input structure, specified 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*.

**Example:**
```
S = [];
S.Exchange = 'Eurex';
S.Product = 'OGBM';
S.ProdType = 'Option';
S.Contract = 'Jan12 P12300';
S.Alias = 'TestInstrument3';
```

**Data Types**
`struct`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
createOrderProfile(X,'Instrument',[],'Customer','<Default>','Alias','','R
```

### 'Property1' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createOrderProfile(X,'Instrument',[],'Customer','<Default>','Alias','','R
```

**Data Types**
`char`

### 'Property2' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createOrderProfile(X,'Instrument',[],'Customer','<Default>','Alias','','R
```

**Data Types**
`char`

## Output Arguments

**P - Order profile**
structure

Order profile, returned as a structure.

## 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` | `createInstrument` | `createNotifier` | `createOrderSet`

## Related Examples


- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9


## Concepts


- "Workflows for Trading Technologies X_TRADER" on page 3-4

## Purpose

Create 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 - Connection object

object structure

`xtrdr` connection object, specified using `xtrdr`.

### S - `xtrdr` input structure with fields

structure

`xtrdr` input structure, specified 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*.

**Example:**
```
S = [];
S.Exchange = 'Eurex';
S.Product = 'OGBM';
S.ProdType = 'Option';
S.Contract = 'Jan12 P12300';
S.Alias = 'TestInstrument3';
```

**Data Types**

`struct`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

**Example:**
```
createOrderSet(X,'Count',0,'Alias','','ReadProperties','b','WriteProperti
```

#### 'Property1' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createOrderSet(X,'Count',0,'Alias','','ReadProperties','b','WriteProperti
```

**Data Types**
`char`

#### 'Property2' - Valid X_TRADER API options
string

Valid X_TRADER API options, specified using the details described in Trading Technologies *X_TRADER API Programming Tutorial* or *X_TRADER API Class Reference*.

**Example:**
```
createOrderSet(X,'Count',0,'Alias','','ReadProperties','b','WriteProperti
```

**Data Types**
`char`

## 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` | `createInstrument` | `createNotifier` | `createOrderProfile`

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

## Purpose

Obtain current X_TRADER data

## Syntax

```
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,F)` returns data for the fields `F` for all instruments associated with the `xtrdr` session object, `X`.

## Input Arguments

### X - Connection object

object structure

`xtrdr` connection object, specified using `xtrdr`.

### S - Instrument object

instrument

Instrument object created by `createInstrument` 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*.

**Example:** `x.Instrument(1)`

### F - Fields for the instrument object

string | cell array of strings

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`.

**Example:** `{'Exchange','Last'}`

**Data Types**
`char | cell`

## Output Arguments

**D - X_TRADER data**
structure

X_TRADER data, returned as a structure

## 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'});
```

```
D =

    Exchange: {'CME'}
        Last: {'45'}
```

### 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'});
```

```
D =

    Exchange: {'CME'}
        Last: {'45'}
```

### 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'});
```

```
D =
```

```
    Exchange: {2x1 cell}
        Last: {2x1 cell}
```

## See Also

`xtrdr` | `createInstrument`

## Related Examples

- "X_TRADER Price Update" on page 4-3
- "X_TRADER Price Update Depth" on page 4-5
- "X_TRADER Order Submission" on page 4-9

## Concepts

- "Workflows for Trading Technologies X_TRADER" on page 3-4

# Index