emsx_api_doc Documentation

Release 2.1.0

Terrence C. Kim

Jun 19, 2023

Table of Contents

This document is for developers who will use the Bloomberg EMSX API to develop custom applications.

The Bloomberg EMSX API is available as desktop API and server-side API. The desktop API requires full Bloomberg terminal to use.

The Bloomberg API uses an event-driven model. The EMSX API is an extension of Bloomberg API 3.0 and it lets users integrate streaming real-time and static data into their own custom applications. The user can choose the data they require down to the level of individual fields. The Bloomberg API 3.0 programming interface implementations are extremely lightweight. For details to the Desktop API, please refer to the Desktop API Programmers Guide from WAPI<GO>.

The Bloomberg API interface is thread-safe and thread-aware, giving applications the ability to utilize multiple processors efficiently. The Bloomberg API supports run-time downloadable schema for the service it provides, and it provides methods to query these schemas at runtime. This means additional service in Bloomberg API is supported without addition to the interface.

The object model for Java, .NET and C++ are identical. The C interface provides a C-style version of the object model.

Important: The Bloomberg EMSX API requires the full understanding of how Bloomberg EMSX<GO> function works within the Bloomberg terminal. Before starting on any EMSX API, please have your local EMSX representative provide a full training of EMSX<GO> function. This documentation does not include any details on how EMSX<GO> works.

Due to the trading nature with the various Trading API's at Bloomberg (e.g. EMSX API, IOI API, etc.) Bloomberg cannot legally assist on the client-side coding other than providing a high-level overview of the service, advice on some of the best practices approach to use the request/response paradigm and asynchronous event-driven nature of the subscription paradigm.

It is highly recommended that the technical resource working on the Trading API has extensive programming experiences and a solid understanding of software application architecture.

Warning: Please note that performance/load test should never be performed on any of the API environment as this is a shared environment and we monitor and increase capacity as needed.

CHAPTER 1

Introduction

The EMSX API is available as programmable and with Excel as both COM and Add-In. The EMSX API provides Bloomberg users with the ability to manage and automate Equities, Futures and Options trading using Microsoft Excel/VBA or creating a custom application in C++, C# (.NET), Python and Java. You can also use Matlab using Trading Toolbox and R.

It also allows users to access the full 2000+ global execution venues available through EMSX.

The EMSX API requires separate authorization by the receiving broker on top of the Bloomberg Authorization.

Note: EMSX API users will need the following steps completed before using the EMSX API at the desktop.

1. Signed ETORSA, Bloomberg Electronic Trading & Order Routing Services Agreement and applicable country legal paperwork, including FIET are required. *An override for UAT testing can be requested in the event clients do not have all legal documentation in place. This cannot be performed for the production environment.*
2. Enable EMSX API per UUID by the Global EMSX Trade Desk for Test (Beta) and Production. Enable Excel Add-In inside the Bloomberg Ribbon for those using the Excel Add-In.
3. To get access to EMSX API in UAT and production, please click <Help><Help> on EMSX<GO>.
4. Download Bloomberg Desktop API v3 SDK from WAPI<GO> in Bloomberg terminal.

For Server Side EMSX API access, the following additional steps are required on top of the desktop EMSX API requirements.

1. Signed EMSxNET Order Originator Agreement.
2. Install serverapi.exe and register with Bloomberg.

To get access to EMSX API in UAT and production, please click <Help><Help> on EMSX<GO>.

1.1 Support

For all EMSX functionality and EMSX API techincal inquires please contact the EMSX Trade Desk. They are available 24/6 and please ensure you provide your Bloomberg UUID.

By Bloomberg Terminal:

HELP 2x (F1 key) on your terminal, ask to speak to the EMSX Trade Desk HELP 1x (F1 key) on your terminal, to compose an email message to the EMSX Trade Desk.

By email:

emsx@bloomberg.net

By Phone:

Please call your local global customer support number and ask to speak to the EMSX Trade Desk

+1-212-617-2000	+44-20-7330-7500	+65-6212-1000

1.2 EMSX API Code Samples

Important: The latest EMSX API Code samples can be found here.

1.3 EMSX API access from Microsoft Excel (COM)

The EMSX API for Excel is accessible using Microsoft Component Object Model (COM) or as part of Bloomberg ribbon within Bloomberg Excel Add-In.

The Microsoft Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software component that can interact with Bloomberg EMSX API services from your desktop where Bloomberg terminal is installed.

ref https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx

1.4 EMSX API access from MATLAB

The EMSX API for MATLAB is accessible by using MATLAB Trading Toolbox in addition to the standard MATLAB package. The matlab samples for EMSX API can be found in both MATLAB Central file exchange.

Please contact your local MATLAB representative for more details on the MATLAB Trading Toolbox.

Important: MathWorks should be your first point of contact for any support while using MATLAB Trading Toolbox. Bloomberg Level II Support desk will not support MATLAB scripts.

1.5 EMSX API access from R

The EMSX API currently can not be accessed via R language. The current R repository is designed for market data Bloomberg API usage using both the subscription and request/response services. Unfortunately, this is not a generic Bloomberg API wrapper for R in its current state.

CHAPTER 2

Server Side EMSX API

EMSX API is available for use via both the desktop (Desktop API, or DAPI) and via a server-side endpoint known as EMSX API Server. The first relies on a logged in Bloomberg terminal for it's connection, whereas the server does not. This makes DAPI unsuitable for mission critical applications.

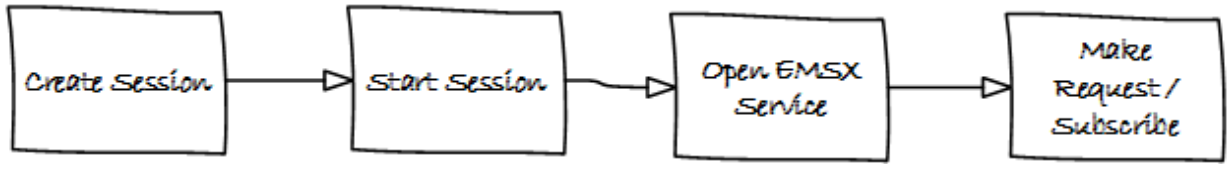
However, the service schema is the same across the two platforms. This means that the code base for an application which was developed on the desktop API is capable of working on the server-side solution without changes to the underlying business logic.

All that is required to move desktop EMSX API applications to the server is the addition of code needed to perform user authentication.

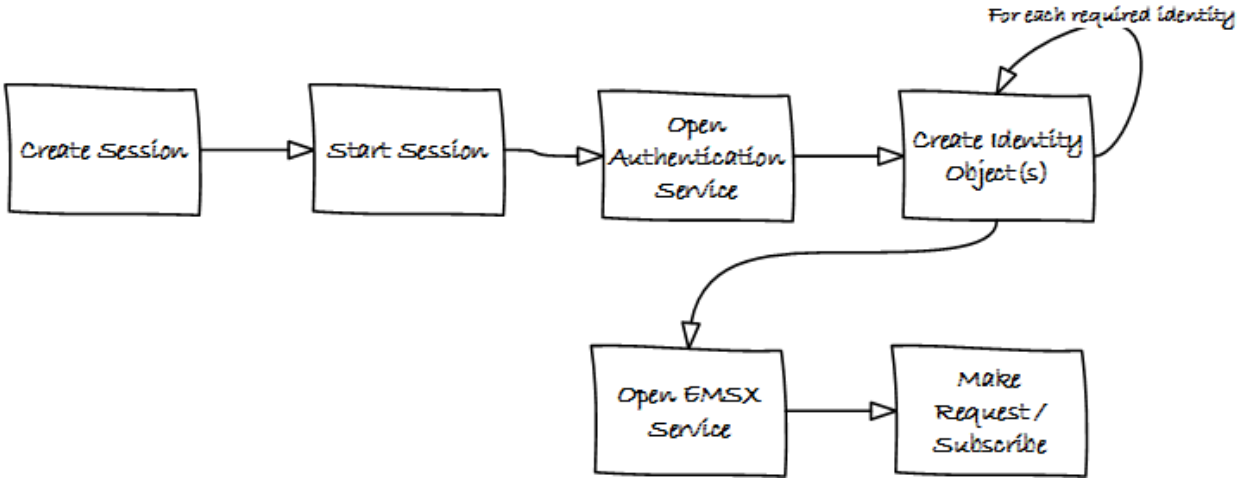
Important: Please reference `BBPC<GO>` in your Bloomberg terminal for full network and connectivity setup. This is in the Bloomberg Transport and Security Specification document.

2.1 Creating User Identities

The steps involved in connecting to the EMSX API on the desktop are as follows:-



In the server environment, the user identities must be created and cached prior to the making requests. Therefore, the process would look as follows:-



Note: Note: The EMSX API Server code samples demonstrate how to create identity object.

The first new step is to open the authentication service. This is done in the same way as for any other service in the Bloomberg API. For example:-

```
d_authsvc = "//blp/apiauth";
session.openServiceAsync(d_authsvc);
```

Once the service is opened, we need to create and send an authorization request. To create an identity for a specific user, you will need the AuthID for the user. This is the name the user is known by in the EMRS system for your server. The values for these names will have been agreed with you as part of the implementation of the server, or subsequently when adding a new user. Also, an IP address is required. The only requirement for this IP address is that it is unique amongst all the identities generated for a session. You can create and send the request as follows:-

```
private Identity userIdentity;

*
*
*

Service authService = session.getService(d_authsvc);
Request authReq = authService.createAuthorizationRequest();

authReq.set("authId", authID);
authReq.set("ipAddress", appIP);

userIdentity = session.createIdentity();

authRequestID = new CorrelationID();

try
{
        session.sendAuthorizationRequest(authReq, userIdentity, authRequestID);
}
catch (Exception e)
{
        System.out.println("Unable to send authorization request: " + e.getMessage());
}
```

In the above code, you can see that an empty identity object is created using `session.createIdentity()`. This is the object that will be populated once successful authentication has been achieved, and it is the object that will need to be cached.

We will receive a Response event for the Authentication service. In the example below, we use a `CorrelationID` to identify messages from the Authentication service, and check for success or failure:-

```
        if(msg.correlationID()==authRequestID) {

        if(msg.messageType().equals(AUTHORIZATION_SUCCESS)) {
                System.out.println("Authorised...Opening EMSX service...");
                System.out.println("Seat Type: " + userIdentity.seatType().
↪toString());
                session.openServiceAsync(d_service);
        } else if(msg.messageType().equals(AUTHORIZATION_FAILURE)) {
                System.out.println("Authorisation failed...");
                System.out.println(msg.toString());
                wait(1000);
                // Automatically retry...
                sendAuthRequest(session);
        } else {
                System.out.println("Unexpected authorisation message...");
                System.out.println(msg.toString());
        }
}
```

When we receive the successful authorization, we can continue with opening the usual EMSX service. If multiple authorization requests have been sent, for a number of different UUIDs, it is necessary to wait for all the responses before being able to use all the identity objects.

In the above code, you will see that we examine the 'seatType' of the identity. The seat type in this case will be either BPS or non-BPS.

2.2 Using User Identities

When a client application connects to `EMSX<GO>` via the API on desktop, it does so by leveraging the identity of the logged in Bloomberg terminal user. This means that when a request or subscription object is received by the Bloomberg infrastructure, the target EMSX blotter can be identified.

In the server environment, there is no Bloomberg terminal, and therefore no implied user can be identified. Moreover, the server is capable of connecting to any number of EMSX user blotters, simultaneously. Therefore, the application making the call must indicate which user is the intended target. This is done through the creation and use of Identity object.

An Identity object represents a specific Bloomberg UUID. Once created, an Identity object can be cached for 24hrs, and used with every `sendRequest()` and `subscribe()` call.

Identity objects are live, that is they remain connected to Bloomberg in real-time and are capable of receiving events. We recommend that an identity is recreated every 24hrs, to ensure that it picks up the latest changes to any user settings, including access to EMSX.

Any number of user Identity object can be created by a server-side application. If the application uses the identities of real traders within a firm, then each trader would have an identity created to represent them in the server application. The server application would, perhaps, receive an instruction from the upstream client-side application to create an order in a trader's blotter. The server application would select the appropriate user identity from the cache and add it to the request.

Migrating the existing desktop application call to a server application simply involves changing all `sendRequest()` and `subscribe()` calls to include the appropriate identity, as follows:-

```
DAPI:
        session.sendRequest(request, requestID);
        session.subscribe(subscriptions);

Server:
        session.sendRequest(request, Identity, requestID);
        session.subscribe(subscriptions, Identity);
```

Following python sample summarizes the above:-

```
import sys
import blpapi
import datetime
import time

SESSION_STARTED                 = blpapi.Name("SessionStarted")
SESSION_TERMINATED              = blpapi.Name("SessionTerminated")
SESSION_STARTUP_FAILURE         = blpapi.Name("SessionStartupFailure")
SESSION_CONNECTION_UP           = blpapi.Name("SessionConnectionUp")
SESSION_CONNECTION_DOWN         = blpapi.Name("SessionConnectionDown")

SERVICE_OPENED                  = blpapi.Name("ServiceOpened")
SERVICE_OPEN_FAILURE            = blpapi.Name("ServiceOpenFailure")

SLOW_CONSUMER_WARNING           = blpapi.Name("SlowConsumerWarning")
SLOW_CONSUMER_WARNING_CLEARED   = blpapi.Name("SlowConsumerWarningCleared")

SUBSCRIPTION_FAILURE            = blpapi.Name("SubscriptionFailure")
SUBSCRIPTION_STARTED            = blpapi.Name("SubscriptionStarted")
SUBSCRIPTION_TERMINATED         = blpapi.Name("SubscriptionTerminated")

AUTHORIZATION_SUCCESS           = blpapi.Name("AuthorizationSuccess")
AUTHORIZATION_FAILURE           = blpapi.Name("AuthorizationFailure")
HANDLE                          = blpapi.Name("handle")


#EMSX/IOI API Server authentication
d_service = "//blp/emapisvc_beta"
d_auth = "//blp/apiauth"
d_host = "1.2.3.4" #static ip address of the server
d_port = 8195
d_user = "MyAuthIDOrEMRSID"


.
.
.

class SessionEventHandler():

        def sendAuthRequest(self, session):

                authService = session.getService(d_auth)
                authReq = authService.createAuthorizationRequest()
                authReq.set("emrsId", d_user)
```

(continues on next page)

(continued from previous page)

```
                authReq.set("ipAddress", d_host)
                self.identity = session.createIdentity ()

                print("Sending authorization request: %s" % (authReq))

                session.sendAuthorizationRequest(authReq, self.identity)

                print("Authorization request.sent.")
.
.
.
    def processSessionStatusEvent(self,event,session):
                print("Processing SESSION_STATUS event")

                for msg in event:

                        print(msg)

                        if msg.messageType() == SESSION_STARTED:
                                print("Session started...")
                                session.openServiceAsync(d_auth)

                        elif msg.messageType() == SESSION_STARTUP_FAILURE:
                                sys.stderr.write("Error: Session startup failed")

                        elif msg.messageType() == SESSION_CONNECTION_UP:
                                print("Session connection is up")

                        elif msg.messageType() == SESSION_CONNECTINO_DOWN:
                                print("Session connection is down")

                        else:
                                print(msg)


        def processServiceStatusEvent(self,event,session):
                print("Processing SERVICE_STATUS event")

                for msg in event:

                        print(msg)

                        if msg.messageType() ==SERVICE_OPENED:

                                serviceName = msg.asElement().getElementAsString(
↪"serviceName");

                                print("Service opened [%s] % (serviceName))

                                if serviceName==d_auth;

                                        print("Auth service opened... Opening␣
↪application service...")
                                        session.openServiceAsync(d_service)

                                elif serviceName==d_service;
```

(continues on next page)

(continued from previous page)

```
                                        print("Application service opened... Sending␣
↪authorization request...")

                                        self.sendAuthRequest(session)

                                elif msg.messageType() == SERVICE_OPEN_FAILURE:
                                        print("Error: Service Failed to open")

        def processAuthorizationStatusEvent(self,event):

                print("Processing AUTHORIZATION_STATUS event")

                for msg in event:

                        print("AUTHORIZATION_STATUS message: %s" % (msg))

.
.
.
```

2.3 Server Side Request/Response

As of today, the following `emapisvc` and `emapisvc_beta` requests are available from the server side access.

Request Name	Action
AssignTrader	Assign an order to another UUID.
CancelRouteEx	Cancel outstanding routes (placements).
CreateOrder	Create an order or stage an order into EMSX<GO>.
CreateOrderAndRouteEx	Create a new order and route in a single request.
CreateOrderAndRouteManually	Create the order and notify EMSX this is routed.
DeleteOrder	Delete an existing order in EMSX<GO>.
GetAllFieldMetaData	Get all field meta data in a response message.
GetBrokerStrategiesWithAssetClass	Get all broker strategy information and asset class data.
GetBrokerStrategyInfoWithAssetClass	Get all broker strategy info and asset class data.
GetBrokerWithAssetClass	Get all broker data with asset class in a response message.
GetFieldMetaData	Get field meta data in a reponse message.
GetTeams	Get team data in a response message.
GroupRouteEx	Submit the entire list as a single route to a basket algorithm.
ModifyOrder	Modify parent order.
ModifyRouteEx	Modify child route.
RouteEx	Route existing order.
RouteManuallyEx	Route manually and notify EMSX that it is routed.

Any other requests will return the following error:

```
"Obsolete request type: " << request_type
```

2.4 How to install serverapi.exe

Please follow the following steps to install and register the installer with Bloomberg Enterprise Solutions with the assistance from EMSX Implementation team.

2.4.1 Linux Environment

The following example is based on the linux environment.

- Run serverapi.exe

```
# ./serverapiinstaller64
```

- You will see the following message

```
# ./serverapiinstaller64
logging to /tmp/bloomberg/install.2019111211.130037.log

Bloomberg ECD Installer for Linux (64-bit)
Version 3.2.2.0


Warning: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of
it, may result in severe civil and criminal penalties, and will be prosecuted
to the maximum extent possible under law.

Would you like to continue? (Y/N):
```

- Type:- Y

```
Would you like to continue? (Y/N): Y

Checking connectivity to Bloomberg ...


a) via Bloomberg Network to [ Hostname = 208.134.161.62 Port = 8194 ] ...
Succeeded.

                                                          [ Hostname = 208.134.161.158␣
↪Port = 8194 ] ...
Succeeded.

                                                          [ Hostname = 208.134.161.18␣
↪Port = 8194 ] ...
Succeeded.

                                                          [ Hostname = 208.134.161.179␣
↪Port = 8194 ] ...
Succeeded.

b) via the Internet to [ Hostname = api1.bloomberg.net Port = 8194 ] ...

Could not resolve host: [ Hostname = api1.bloomberg.net Port = 8194 ]
```

(continues on next page)

(continued from previous page)

```
Error.

                                        [ Hostname = api2.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api2.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api3.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api3.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api4.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api4.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api5.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api5.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api6.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api6.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api7.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api7.bloomberg.net Port = 8194 ]

Error.

                                        [ Hostname = api8.bloomberg.net Port = 8194 ]␣
↪...

Could not resolve host: [ Hostname = api8.bloomberg.net Port = 8194 ]

Error.

Internet connectivity unavailable. Connecting via the Bloomberg Network.
```

- Select the appropriate network option if it doesn't select by default (private vs. public/internet)

```
Select Product Class
1) blpddm    Software that provides development access to distribute data locally or␣
↪contribute data to Bloomberg.
```

(continues on next page)

(continued from previous page)

```
2) ServerApi Provides access to Bloomberg real-time streaming and static data
0) Quit
```

- Select:- 2 for Server API

```
Please enter selection: 2

Installation path:
                '/opt/local'

Use this path? (Y/N/Q):
```

- Select:- York

```
Creating the root directory /opt/local ...
done.

Downloading latest installer ...
done.

logging to /tmp/bloomberg/install.2019111211.130037.log

Beginning new install ...
```

Note: If the default port is already being used by a different service it may show the following message:

```
*** WARNING: Port conflict detected with other service.
The port of the Desktop will conflict with the ServerApi should installation proceed.␣
↪If you still want to install
ServerApi, you will need to specify a different port number.

 Do you want to continue with the installation? (Y/N) [N]:
```

- Select:- Y and enter the port

```
Do you want to continue with the installation? (Y/N) [N]:y
Please enter ServerApi listen port: [8294]:8294
```

- Select the version:-

```
Versions available for ServerApi
1) 3.86.5.1     Linux64     ServerAPI 2017-06

2) 3.88.0.1     Linux64     ServerAPI 2017-08

3) 3.90.3.1     Linux64     ServerAPI 2017-10

4) 3.90.6.1     Linux64     ServerAPI 2018-01

5) 3.98.5.1     Linux64     ServerAPI 2018-04

6) 3.102.0.1    Linux64     ServerAPI 2018-05

7) 3.106.0.1    Linux64     ServerAPI 2018-07
```

(continues on next page)

(continued from previous page)

```
8) 3.112.3.1    Linux64     ServerAPI 2018-10

9) 3.112.4.1    Linux64     ServerAPI 2019-01

10) 3.114.9.1    Linux64     ServerAPI 2019-04

11) 3.118.9.1    Linux64     ServerAPI 2019-07

12) 3.120.2.0    Linux64     Development B-Pipe 2019-10 (64-bit)

13) 3.120.2.1    Linux64     ServerAPI 2019-10

0) Quit
Please enter version of ServerApi that you want to install:
```

- Select the latest:-

```
Please enter version of ServerApi that you want to install: 13
Downloading ServerApi components ...
```

- Enter other information:-

```
Enter the following information:

                Country (e.g., USA):
                State (e.g., NY):
                City or Town (e.g., New York):
                Company Name (e.g., Bloomberg L.P.):
                Department Name (e.g., Equity Trading)
```

- Finished:-

```
Enter the following information:

                Country (e.g., USA): USA
                State (e.g., NY): NY
                City or Town (e.g., New York): New York
                Company Name (e.g., Bloomberg L.P.): My Firm
                Department Name (e.g., Equity Trading): Futures Trading

Creating certificate ...
done.

Registering server ...
done.

done.


Call Bloomberg's Global Customer Support at +1 (212) 318-2000 and ask for the Global␣
↪Installs desk. The Bloomberg representative will ask you to read your registration␣
↪number over the phone four characters at a time.


Your registration key is:
                123b-4567-1ab2-12c9-g66f-964e-h50b-fa48-c78t-a123
```

(continues on next page)

(continued from previous page)

```
This key was also saved in regkey.txt in the ServerApi root directory.

ServerApi installation completed. Press ENTER to quit:
```

Note: Once the registration process is completed. EMSX Implementation team globally will assist with configuring the Server Side EMSX API with various execution destinations per client request.

2.4.2 Windows Environment

The following example is based on the windows environment.

- Run serverapi.exe

```
C:\temp>serverapiinstaller.exe
```

- You will see the following message

```
C:\temp>serverapiinstaller.exe
logging to C:\temp\install.2016102610.152444.log

Bloomberg ECD Installer for Windows (32-bit)
Version 3.2.2.0


Warning: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of
it, may result in severe civil and criminal penalties, and will be prosecuted
to the maximum extent possible under law.


logging to C:\temp\install.2016102610.152444.log

Bloomberg ECD Installer for Windows (32-bit)
Version 3.2.2.0


Warning: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of
it, may result in severe civil and criminal penalties, and will be prosecuted
to the maximum extent possible under law.


Would you like to continue? (Y/N):
```

- Type:- Y

```
Would you like to continue? (Y/N): y

Checking connectivity to Bloomberg ...
```

(continues on next page)

(continued from previous page)

```
a) via Bloomberg Network to [ Hostname = 208.134.161.62 Port = 8194 ] ...
Succeeded.

                            [ Hostname = 208.134.161.158 Port = 8194 ] ...
Succeeded.

                            [ Hostname = 208.134.161.18 Port = 8194 ] ...
Succeeded.

                            [ Hostname = 208.134.161.179 Port = 8194 ] ...
Succeeded.

b) via the Internet to [ Hostname = api1.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api2.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api3.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api4.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api5.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api6.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api7.bloomberg.net Port = 8194 ] ...
Succeeded.

                        [ Hostname = api8.bloomberg.net Port = 8194 ] ...
Succeeded.



Which of the above routes will you use to connect to Bloomberg? (a/b):
```

- Select the appropriate network option (private vs. public/internet)

```
Which of the above routes will you use to connect to Bloomberg? (a/b): b

Bloomberg Network connectivity unavailable. Connecting via the Internet.

Select Product Class
1) blpddm    Software that provides development access to distribute data locally or␣
↪contribute data to Bloomberg.
2) ServerApi Provides access to Bloomberg real-time streaming and static data
0) Quit

Please enter selection:
```

- Select:- 2 for Server API

```
Please enter selection: 2

Installation path:
        'C:\'

Use this path? (Y/N/Q):
```

- Select:- Y

```
Use this path? (Y/N/Q): y

Downloading latest installer ...
done.

logging to C:\temp\install.2016102610.152444.log

Beginning new install ...
```

Note: If the default port is already being used by a different service it may show the following message:

```
*** WARNING: Port conflict detected with other service.
The port of the Desktop will conflict with the ServerApi should installation proceed.␣
↪If you still want to install
ServerApi, you will need to specify a different port number.

 Do you want to continue with the installation? (Y/N) [N]:
```

- Select:- Y and enter the port

```
Do you want to continue with the installation? (Y/N) [N]:y
Please enter ServerApi listen port: [8294]:8294
```

- Select the version:-

```
Versions available for ServerApi
1) 3.46.6.0     Windows     ServerAPI 2014-07

2) 3.48.8.1     Windows     ServerAPI 2014-09

3) 3.48.9.1     Windows     ServerAPI 2014-11

4) 3.50.7.1     Windows     ServerAPI 2015-01

5) 3.56.4.1     Windows     ServerAPI 2015-04

6) 3.60.0.1     Windows     ServerAPI 2015-07

7) 3.64.5.1     Windows     ServerAPI 2015-10

8) 3.70.0.1     Windows     ServerAPI 2016-01

9) 3.72.2.1     Windows     ServerAPI 2016-04

10) 3.82.3.1     Windows     ServerAPI 2016-10
```

(continues on next page)

(continued from previous page)

```
11) 3.46.6.0     Windows64   ServerAPI 2014-07

12) 3.48.8.1     Windows64   ServerAPI 2014-09

13) 3.48.9.1     Windows64   ServerAPI 2014-11

14) 3.50.7.1     Windows64   ServerAPI 2015-01

15) 3.56.4.1     Windows64   ServerAPI 2015-04

16) 3.60.0.1     Windows64   ServerAPI 2015-07

17) 3.64.5.1     Windows64   ServerAPI 2015-10

18) 3.70.0.1     Windows64   ServerAPI 2016-01

19) 3.72.2.1     Windows64   ServerAPI 2016-04

20) 3.82.3.1     Windows64   ServerAPI 2016-10

0) Quit
Please enter version of ServerApi that you want to install:
```

- Select the latest:-

```
Please enter version of ServerApi that you want to install: 20
Downloading ServerApi components ...
```

- Enter other information:-

  Enter the following information:

  Country (e.g., USA): State (e.g., NY): City or Town (e.g., New York): Company Name (e.g., Bloomberg L.P.): Department Name (e.g., Equity Trading):

- Finished:-

```
Enter the following information:

Country (e.g., USA): USA
State (e.g., NY): NY
City or Town (e.g., New York): New York
Company Name (e.g., Bloomberg L.P.): Bloomberg LP
Department Name (e.g., Equity Trading): EMSX

Creating certificate ...
done.

Registering server ...
done.


Do you want to install ServerApi as a Windows Service? (Y/N): y

Installing ServerApi as a windows Service...
service ServerApi configured for restart on first error
 done
```

(continues on next page)

(continued from previous page)

```
done.


*** Please reboot your computer for changes to take effect ***


Call Bloomberg's Global Customer Support at +1 (212) 318-2000 and ask for the
Global Installs desk. The Bloomberg representative will ask you to read your
registration number over the phone four characters at a time.


Your registration key is:
        321c-5ad5-7fa8-2954-1930-abb0-b64c-ecaf-1505-64d4
```

Note: Once the registration process is completed. EMSX Implementation team globally will assist with configuring the Server Side EMSX API with various execution destinations per client request.

CHAPTER 3

Programmable EMSX API

The programmable API provides developers with access to EMSX data via a number of programming languages. It can be used independantly of the EMSX Excel add-in, or as a complement. The API provide the developer with the means to replicate most of the behaviour available from the EMSX<GO> in the terminal.

The API supports two distinct programming paradigms; Subscription and Request/Response. Anyone already familiar with the Bloomberg API will recognize this approach. The EMSX API is simply an additional service (`//blp/emapisvc` or `//blp/emapisvc_beta`) on the Bloomberg API, with certain subtle differences due to the nature of the data involved.

The **Request/Response** methods are used to directly affect the state of the order book. Using these methods, the developer can Create and Delete (or Cancel) orders and routes (placements). When a request is made, for example `CreateOrder`, the application must supply the necessary field values as parameters. The application must then wait for, and process, any responses (success or failure, for example) before the order or route can be futher utilized. Requests are matched to their responses through the use of CorrelationIDs.

The **subscription** service is used to maintain a local view of a user's order book. Subscriptions are made for either orders or routes (placements), and any number of subscriptions can be made. The subscription is made at a user level, meaning all orders (or routes) for a given user are monitored on single subscription. When implementing subscription service, it's important to write the code using two separate `.subscribe()` events for the order and route subscriptions.

When a subscription is first made, the application will receive all the necessary messages to bring the local image of the user's EMSX order book up to date. These initial messages will contain all the relevant fields for each order, both static and dynamic. Thereafter (within the same session), the user will only receive dynamic fields in any update messages. It is the developer's responsibility to identify the changes, and respond appropriately. These messages are not stateful, and the API does not guarentee the order in which messages are received. However, this should not negatively impact the application, as long as the developer is aware of this and takes it into account.

For example, when a `CreateOrder` is issued, as discussed above, it is perfectly feasible for a subscription event to be received before the response to the request. As this is a new order, the `EMSX_SEQUENCE` (the order ID number) will not yet be known on the client side. Therefore, you may be receiving messages for a sequence number that is not recognised, and will not be known until the response to the original `CreateOrder` request is processed. This can be dealt with through simple buffering of the subscription events. In order to simplify this process, non AIM users have the option of using `EMSX_ORD_REF_ID` in subscription by supplying the `EMSX_ORDER_REF_ID` in the Request. This will allow the user to use the subscription event without having to wait for the response. The user can

match requests with responses as well as subscription events. EMSX_ORDER_REF_ID has 16 character limitations but otherwise should be good to use as custom user defined field.

The EMSX_REQUEST_SEQ should also be added to every request. The EMSX_REQUEST_SEQ should consist of 64-bit integer and should be reset once a week. The purpose of this unique user assigned sequence number is to prevent duplicate requests from being sent during system outages. The number also should be unique per serial number of the Bloomberg terminal.

3.1 EMSX Features

The EMSX API supports 99.9% of the features supported in EMSX<GO> function.

The following standalone EMSX settings will also impact the EMSX API.

Important: Please note the following EMSX settings are changed by Bloomberg at the user or user firms request.

EMSX Setting
EMSX Routing enabled
Orders=routes enabled
Staging Protection enabled
Use B/O & S/C for Futures enabled
Allow MKT routes on LMT
Broker (Hard) Restrictions
Directed Broker
Restricted Secuirites List (EMSX)
Cross check for Equity
Cross check for Futures
Broker (Soft) restriction
Allow After-Market Routing for Day Order
Exec/Research/Risk Capital Rate Type
Enable Route as Futures Spread
Enable Basket All or None
Enable Restricted Securities Validation (RTIP)
Filter out Directed/Restricted brokers
Enable Team Risk Ticket
Enable Routing InvestorID to Broker
Allow Blottery Snc Orders to be Deleted
Enable Centralized Trading controls
Block Market Routes
AIM: Use Settlement Date from B/S
AIM: Send AIM Order# in BlockID tag
LMSA: Lock Broker Code on Order
LMSA: AIM Restricted Order Violation Setting
LMSA: AIM Add Order from EMSX Lanuch Ticket

Important: The following settings are controlled by the user of EMSX<GO>.

EMSX User Defaults	EMSX Setting Location
Home Currency	Setting under Confirmation & Warnings in EMSX
Warn About Restricted Short Sells	Setting under Confirmation & Warnings in EMSX
Order Violation Settings	Setting under Confirmation & Warnings in EMSX
Confirm Order Violation	Setting under Confirmation & Warnings in EMSX
Quantity Warning	Setting under Confirmation & Warnings in EMSX
Quantity Maximum	Setting under Confirmation & Warnings in EMSX
Market Value Warning	Setting under Confirmation & Warnings in EMSX
%ADV Warning Threshold	Setting under Confirmation & Warnings in EMSX
%ADV Maximum Threshold	Setting under Confirmation & Warnings in EMSX
ADV Benchmark	Setting under Confirmation & Warnings in EMSX
Price Tolerance level	Setting under Confirmation & Warnings in EMSX

EMSX Routing Defaults	EMSX Setting Location
Strategy Time Zone	Setting under Routing Generic
Show Commission Fields	Setting under Routing Generic
Show Basket Name in Broker Notes	Setting under Routing Generic
Send Odd Lots	Setting under Routing Generic
Send Parent Order Instruction*	Setting under Routing Generic
User Order Values for Routing	Setting under Routing Generic

3.2 EMSX Teams

The EMSX API allows the same action on `TEAMVIEW` as you would have permission on `EMSX<GO>` function.

The `TEAMVIEW` feature in `EMSX<GO>` allows a team member to view or take action on behalf of the team members based on the team setting within `EMSX<GO>`.

For EMSX API, This offers flexibilities within the application design. For example, a single subscription with the team name can capture all the events for the team members. The topic string for using team remains the same as non-team with the exception of adding team name on the topic string as illustrated below.

Important: //blp/emapisvc_beta/order;team=my_team_name?fields=EMSX_ASSIGNED_TRADER, EMSX_BASKET_NAME, EMSX_CFD_FLAG, EMSX_AMOUNT

Trading on behalf of team members from `TEAMVIEW` requires creating a route on behalf of the team member. The service object of type `RouteEx` and fill in the required fields before submitting the request.

Within `RouteEx`, there is an element `EMSX_TRADER_UUID` where the user can enter the order owner's Bloomberg UUID. Bloomberg will do the validation against the user privilege setup via `EMT<GO>` and `EMBR<GO>`.

A user can be part of more than one team on the backend. When the user creates the topic string and does not belong to a team or specify a wrong team, the user will receive an error.

In cases where a user is defined as a member of multiple teams, then the user will need to supply multiple subscriptions. (One for each team). These subscriptions should be monitored separately since the user will receive two notifications.

Important: It's best to keep the overall design simple. TEAM is a heirarchical structure and thus best to have a single order and single route subscription for the entire TEAM strucuture and avoid replication. The replication increases the bandwidth usage and provides ZERO benefit for the end client.

3.3 EMSX Element Definitions

For information on accessing field meta data, this is currently only supported within Bloomberg terminal.

The user will need to access `FLDS<GO>` function within the Bloomberg terminal. Once in `FLDS<GO>`, type EMSX underneath the security section and choose EMSX under the filter. The source is Calcrt and should select All for Field Type.

3.4 EMSX Element Definition (A to M)

The EMSX element definitions will include the type of the element and will inform whether the element is an `ORDER`, `ROUTE`, or sometimes both `O,R` elements. The type consists of `INT64`, `INT32`, `STRING`, and `FLOAT64`.

Field	Definition
API_SEQ_NUM	INT64 Special field to indicate the sequence number of the API events. The number begins at 1 and increases with each event posted to a client subscription. It can be used by the client side to guarantee order, and to identify any gaps in subscription events.
EMSX_ACCOUNT	STRING O,R The account of the routing firm as designated by the broker chosen. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_AMOUNT	INT32 O,R The total amount of the order or route. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_APA_MIC	STRING ROUTE Approved publication arrangement in MiFID II. This is a route level field.
EMSX_ASSET_CLASS	STRING STATIC ORDER The asset class of the order. This field is applicable to trades on an order level, and does not populate on a per security basis. This is a static field.
EMSX_ASSIGNED_TRADER	STRING ORDER The name of the trader assigned to the order. This field is applicable to trades on an order level, and does not populate on a per security basis.
EMSX_AVG_PRICE	FLOAT64 O,R The average price for one share executed with the order, calculated over the life of the order. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.

EMSX_CLIENT_IDENTIFICATION	STRING O,R MiFID II field for client Identification.

EMSX_CLIENT_ORDER_ID	STRING ROUTE The client order ID identifier generated between EMSX and the EOR Broker. This value is unique per day. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_COMM_DIFF_FLAG	STRING O,R The EMSX Commission Difference between broker commission and AIM (Asset and Investment Manager) commission values. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_COMM_RATE	FLOAT64 O,R The EMSX Commission Rate of commission charged on the trade. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_CURRENCY_PAIR	STRING STATIC O,R The EMSX Currency Pair which provides the spot rate to convert the security's currency and the user's currency. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_CUSTOM_ACCOUNT	STRING ROUTE The EMSX Route Account, is the account value at the level of the route. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_CUSTOM_NOTEn	STRING ORDER 79-character free text field.
EMSX_DATE	INT32 ORDER The EMSX Order Creation Date is the date on which the order is created. This field is applicable to trades on an order level, and does not populate on a per security

EMSX_LAST_FILL_TIME_MICROSEC	INT32 ROUTE The last fill time based on the user's time zone in microseconds. This field is applicable to trades on a route level, and does not populate on a per security basis.

EMSX_LAST_MARKET	STRING ROUTE The last market of execution for a trade as returned by the broker.This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_LAST_PRICE	FLOAT64 ROUTE The last execution price for a trade. This field is applicable to trades on a route level, and does not populate ona per security basis.
EMSX_LAST_SHARES	INT32 ROUTE The last executed quantity for a trade. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_LEG_FILL_DATE_ADDED	INT32 ROUTE The date added for the leg fill.
EMSX_LEG_FILL_PRICE	FLOAT64 ROUTE The leg fill price.
EMSX_LEG_FILL_SEQ_NO	INT32 ROUTE The leg fill sequence number.
EMSX_LEG_FILL_SHARES	FLOAT64````ROUTE The leg fill shares.
EMSX_LEG_FILL_SIDE	STRING ROUTE The leg fill side.
EMSX_LEG_FILL_TICKER	STRING ROUTE The leg fill ticker.
EMSX_LEG_FILL_TIME_ADDED	INT32 ROUTE The time added for the leg fill.
EMSX_LIMIT_PRICE	FLOAT64 O,R The price which is the maximum the order to buy securities or commodities should be executed at; or the minimum at which securities or commodities should be sold. This field is applicable to trades on an order and/or route level, and does not

3.5 Multi-Leg Element Definition

Field	Definition
EMSX_ML_ID	STRING ROUTE The multi-leg ID.
EMSX_ML_LEG_QUANTITY	INT32 ROUTE The EMSX Multi-Leg Shares per Leg is the number of shares per leg in the multi-leg strategy. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ML_NUM_LEGS	INT32 ROUTE The EMSX Multi-Leg Number Legs is the number of legs in the multi-leg strategy. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ML_PERCENT_FILLED	FLOAT64 ROUTE The EMSX Multi-Leg Percent Filled is the percent of legs filled in a multi-leg strategy. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ML_RATIO	FLOAT64 ROUTE The EMSX Multi-Leg Ratio is the factor that controls the number of securities in each leg. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ML_REMAIN_BALANCE	FLOAT64 ROUTE The EMSX Multi-Leg Remaining Balance is the balance yet to be filled across the legs of a multi-leg strategy. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ML_STRATEGY	STRING ROUTE The EMSX Multi-Leg Strategy Name is the name of the multi-leg strategy for the route. This field is applicable to

3.6 EMSX Element Definition (N to Z)

Field	Definition
EMSX_NOTES	STRING O,R The EMSX Instructions is the free form instructions that may be sent to the broker. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_NSE_AVG_PRICE	FLOAT64 O,R The EMSX National Stock Exchange Average Price is the average price of the fills completed for the order or route on the National Stock Exchange (NSE). This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_NSE_FILLED	INT32 O,R The EMSX National Stock Exchange Filled is the total quantity of the fills completed for the order or route on the National Stock Exchange (NSE). This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_ORD_REF_ID	STRING ORDER The EMSX Order Reference ID. The element is called the EMSX_ORDER_REF_ID in the request/response services. Not available to AIM users.
EMSX_ORDER_AS_OF_DATE	INT32 ORDER The order as of date in EMSX in New York time zone.

EMSX_ORDER_AS_OF_TIME_MICROSEC	FLOAT64 ORDER The order as of time in microseconds in New York time zone.

EMSX_ORDER_TYPE	STRING O,R The order type in EMSX. (e.g. market, limit, stop limit and etc.)
EMSX_ORIGINATE_TRADER	STRING ORDER The trader who routed the order. This field is applicable to trades on an order level, and does not populate on a per security basis.
EMSX_ORIGINATE_TRADER_FIRM	STRING STATIC ORDER The firm of the trader who routed the order. This field is applicable to trades on an order level and does not populate on a per security basis.
EMSX_OTC_FLAG	STRING ROUTE The OTC flag in EMSX.
EMSX_P_A	STRING ROUTE The EMSX Principal/Agency element specifies the capacity in which the broker acts for a particular order and route; either 'Principal' or 'Agency'. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_PERCENT_REMAIN	FLOAT64 O,R The remaining balance of the order as a percentage of the projected remaining volume in the day. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_PM_UUID	INT32 STATIC ORDER The Portfolio Manager UUID in AIM.
EMSX_PORT_MGR	STRING STATIC ORDER The EMSX Portfolio Manager is the name of the portfolio manager in the AIM function. For standalone users, this is the same as the EMSX Trader Name. This field is applicable to trades on an order level, and does not

EMSX_ROUTE_AS_OF_TIME_MICROSEC	FLOAT64 ROUTE The route as of time in microseconds, in New York time zone.
EMSX_ROUTE_CREATE_DATE	INT32 STATIC ROUTE The date of the creation of the route in the user's time zone. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ROUTE_CREATE_TIME	INT32 STATIC ROUTE The time of the creation of the route in seconds from midnight in the user's time zone. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_ROUTE_CREATE_TIME_MICROSEC	FLOAT64 STATIC ROUTE EMSX_ROUTE_CREATE_TIME in microseconds.
EMSX_ROUTE_ID	INT32 STATIC O,R The transaction number of the route in the system. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_ROUTE_LAST_UPDATE_TIME	INT32 ROUTE The time stamp of the last execution or cancellation on a route. This field is applicable to trades on a route level and does not populate on a per security basis.

EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC	FLOAT64 ROUTE EMSX_ROUTE_LAST_UPDATE_TIME in microseconds.

EMSX_ROUTE_PRICE	FLOAT64 O,R The route price benchmark for the route. This is the midpoint during market hours, and the next opening price between exchange sessions. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_ROUTE_REF_ID	STRING ROUTE The EMSX Route Reference ID. The element is called the EMSX_ROUTE_REF_ID in the request/response services. Not available to AIM users.
EMSX_SEC_NAME	STRING STATIC ORDER The EMSX Security Name is the long name of the security being traded in EMSX. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_SEDOL	STRING STATIC ORDER The EMSX Stock Exchange Daily Official List - SEDOL (Stock Exchange Daily Official List) number of the security in the order. This field is applicable to trades on an order level and does not populate on a per security basis.
EMSX_SEQUENCE	INT32 STATIC O,R The sequence number generated by the EMSX function for the order. This field is applicable to trades on an order and/or route level,and does not populate on a per security basis.
EMSX_SETTLE_AMOUNT	FLOAT64 O,R The EMSX Net Money is the executed value of trade net of commission, taxes, and fees. This field is applicable to trades on an order and/or route level, and does not populate on

EMSX_TRADE_REPORTING_INDICATOR	STRING STATIC ORDER The trade reporting indicator for MiFID II.

EMSX_TRADER	STRING ORDER The current trader's Bloomberg login name. This field is to trades on an order level, and does not populate on a per security basis.
EMSX_TRADER_NOTES	STRING ORDER The free form notes for the trader which are not passed on to the brokers. This field is applicable to trades on an order level, and does not populate on a per security basis.

EMSX_TRANSACTION_REPORTING_MIC	STRING ROUTE The transaction reporting MIC code in MiFID II.

EMSX_TS_ORDNUM	INT32 STATIC ORDER The order number generated by the AIM. For a non-AIM user, this number is the same as the EMSX_SEQUENCE Number. This field is applicable to trades on an order level, and does not populate on a per security basis.
EMSX_TYPE	STRING O,R The type of the order; this can be a preconfigured valued or a value configured by the individual broker. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_UNDERLYING_TICKER	STRING STATIC ORDER The instrument to which a derivative, such as an equity or index option, is related. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_URGENCY_LEVEL	INT32 ROUTE The integer which is the parameter for a route strategy, which determines a route's priority. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_USER_COMM_AMOUNT	FLOAT64 O,R The EMSX User Commission Amount is the total commission charged on the trade based on user-defined commission rates entered. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.
EMSX_USER_COMM_RATE	FLOAT64 O,R The EMSX User Commission Rate is the user-defined commission rate for the trade. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.

3.7 Accessing the Test Environment

Bloomberg provides a test environment for clients to build and test their strategies using the EMSX API.

This is accomplished by referencing `//blp/emapisvc_beta` as the service name in your program. This command will allow your service to redirect all EMSX API requests and subscriptions to the test environment.

Once the client has thoroughly tested the custom-built strategies, they can access the production environment by changing the service name from `//blp/emapisvc_beta` to `//blp/emapisvc`.

Inside the Bloomberg Terminal type UAT ON <GO>.This command allows the particular terminal window and launchpad to log into the beta environment. Please note, when a user is remote into the beta environment it only affects that particular terminal window and the other Bloomberg panels will not be affected by the UAT ON <GO> command.

To check which environment your current view is in, type VSAT <GO> inside the Bloomberg terminal.

To get back to production type UAT OFF <GO>. Please note that the testing environment in Beta will not operate in the exact same way as the production environment. Also, please note that the beta environment is a lot slower than the production environment and no one should perform any volume or load testing in the beta environment.

3.8 API Demo Tool

API Demo Tool is a handy tool while developing on any Bloomberg API services. The API Demo Tool provides real-time schema viewing tool among other handy tools that can be leveraged during the initial development.

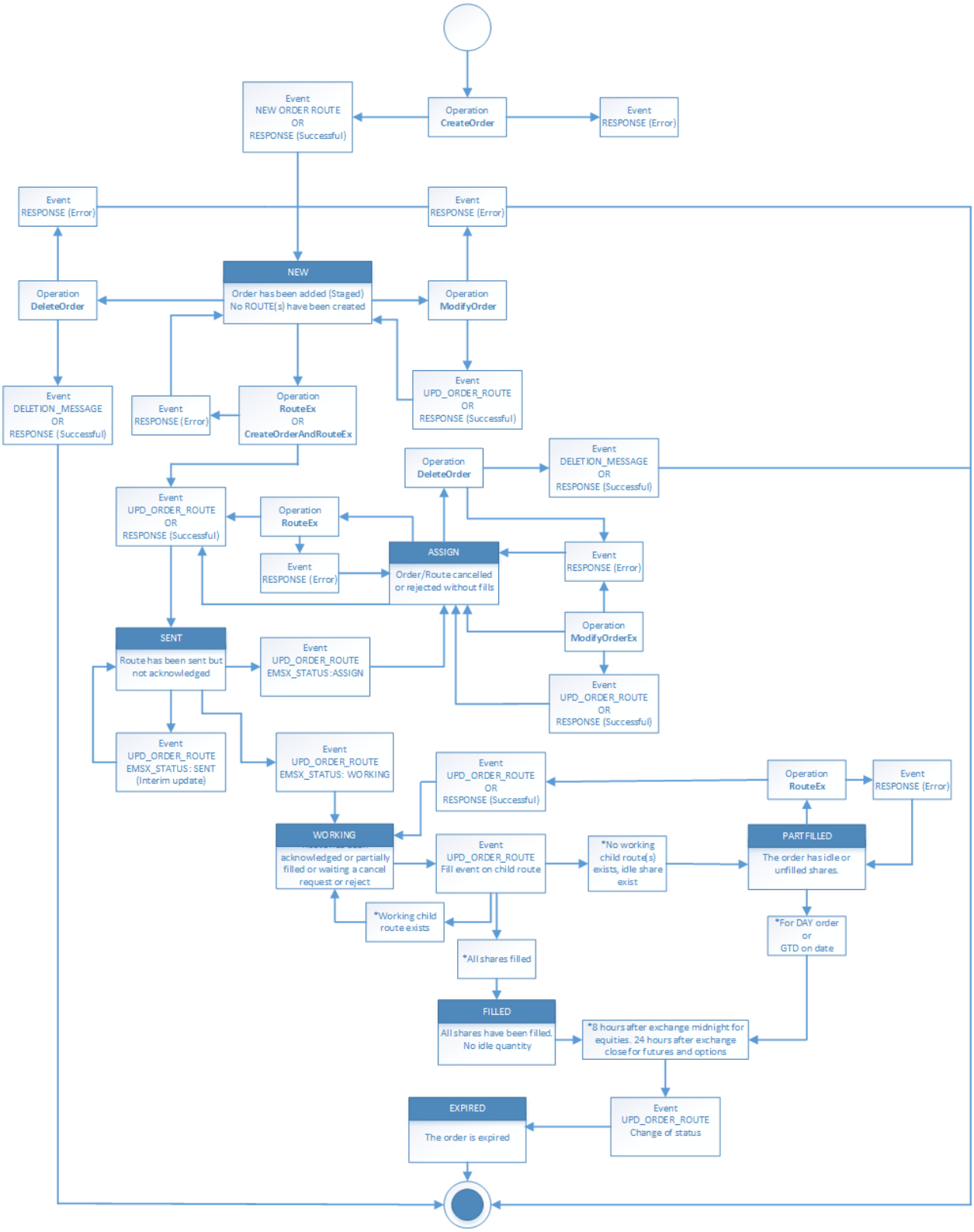
The API Demo Tool can be downloaded from the Bloomberg terminal along with other generic Bloomberg API code samples.

> `WAPI<GO>` >> API Download Center >> Download

3.9 Order State Diagram

Following is an order state diagram for EMSX API:-

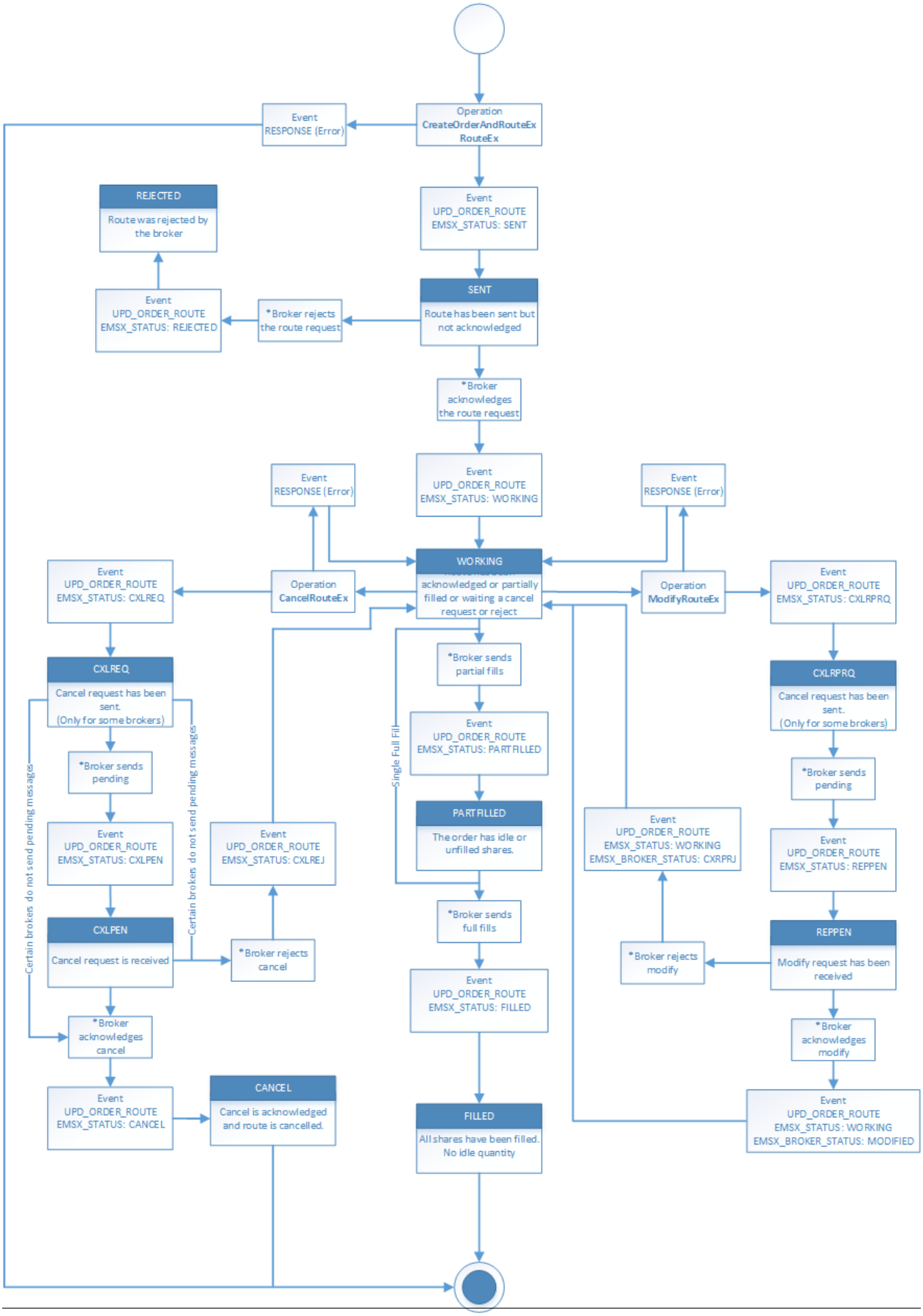
Order State PDF


Event
NEW ORDER ROUTE
OR
RESPONSE (Successful)
Operation
CreateOrder
Event
RESPONSE (Error)
Event
RESPONSE (Error)
Event
RESPONSE (Error)
NEW
Order has been added (Staged)
No ROUTE(s) have been created
Operation
DeleteOrder
Operation
ModifyOrder
Event
DELETION_MESSAGE
OR
RESPONSE (Successful)
Event
RESPONSE (Error)
Operation
RouteEx
OR
CreateOrderAndRouteEx
Event
UPD_ORDER_ROUTE
OR
RESPONSE (Successful)
Operation
DeleteOrder
Event
DELETION_MESSAGE
OR
RESPONSE (Successful)
Event
UPD_ORDER_ROUTE
OR
RESPONSE (Successful)
Operation
RouteEx
Event
RESPONSE (Error)
ASSIGN
Order/Route cancelled
or rejected without fills
Event
RESPONSE (Error)
Operation
ModifyOrderEx
SENT
Route has been sent but
not acknowledged
Event
UPD_ORDER_ROUTE
EMSX_STATUS:ASSIGN
Event
UPD_ORDER_ROUTE
OR
RESPONSE (Successful)
Event
UPD_ORDER_ROUTE
EMSX_STATUS: SENT
(Interim update)
Event
UPD_ORDER_ROUTE
EMSX_STATUS: WORKING
Event
UPD_ORDER_ROUTE
OR
RESPONSE (Successful)
Operation
RouteEx
Event
RESPONSE (Error)
WORKING
acknowledged or partially
filled or waiting a cancel
request or reject
Event
UPD_ORDER_ROUTE
Fill event on child route
*No working
child route(s)
exists, idle share
exist
PARTFILLED
The order has idle or
unfilled shares.
*Working child
route exists
*For DAY order
or
GTD on date
*All shares filled
FILLED
All shares have been filled.
No idle quantity
*8 hours after exchange midnight for
equities. 24 hours after exchange
close for futures and options
EXPIRED
The order is expired
Event
UPD_ORDER_ROUTE
Change of status

3.10 Route State Diagram

Following is a route state diagram for EMSX API:-

Route State PDF


Operation
CreateOrderAndRouteEx
RouteEx
Event
RESPONSE (Error)
Event
UPD_ORDER_ROUTE
EMSX_STATUS: SENT
SENT
Route has been sent but not acknowledged
*Broker rejects the route request
Event
UPD_ORDER_ROUTE
EMSX_STATUS: REJECTED
REJECTED
Route was rejected by the broker
*Broker acknowledges the route request
Event
UPD_ORDER_ROUTE
EMSX_STATUS: WORKING
Event
RESPONSE (Error)
Event
RESPONSE (Error)
WORKING
acknowledged or partially filled or waiting a cancel request or reject
Operation
CancelRouteEx
Operation
ModifyRouteEx
Event
UPD_ORDER_ROUTE
EMSX_STATUS: CXLREQ
Event
UPD_ORDER_ROUTE
EMSX_STATUS: CXLRPRQ
CXLREQ
Cancel request has been sent.
(Only for some brokers)
CXLRPRQ
Cancel request has been sent.
(Only for some brokers)
*Broker sends pending
*Broker sends pending
Event
UPD_ORDER_ROUTE
EMSX_STATUS: CXLPEN
Event
UPD_ORDER_ROUTE
EMSX_STATUS: REPPEN
CXLPEN
Cancel request is received
REPPEN
Modify request has been received
Certain brokers do not send pending messages
Certain brokers do not send pending messages
Event
UPD_ORDER_ROUTE
EMSX_STATUS: CXLREJ
*Broker rejects cancel
*Broker rejects modify
Event
UPD_ORDER_ROUTE
EMSX_STATUS: WORKING
EMSX_BROKER_STATUS: CXRPRJ
*Broker acknowledges cancel
*Broker acknowledges modify
Event
UPD_ORDER_ROUTE
EMSX_STATUS: CANCEL
CANCEL
Cancel is acknowledged and route is cancelled.
Event
UPD_ORDER_ROUTE
EMSX_STATUS: WORKING
EMSX_BROKER_STATUS: MODIFIED
*Broker sends partial fills
Single Full Fill
Event
UPD_ORDER_ROUTE
EMSX_STATUS: PARTFILLED
PARTFILLED
The order has idle or unfilled shares.
*Broker sends full fills
Event
UPD_ORDER_ROUTE
EMSX_STATUS: FILLED
FILLED
All shares have been filled.
No idle quantity

3.11 EMSX API Schema

EMSX API Schema

3.12 EMSX API History Service Schema

EMSX API History Service Schema

3.13 Session Object

Connecting and creating a session object for EMSX API uses BBCOMM for desktop and AuthID or EMRSID + IP Address for server side EMSX API access.

BBCOMM is the service that runs on an EMSX user's computer and conducts all communication to and from Bloomberg. The application connects to the local BBCOMM and the most common configuration is "localhost" for hostname and "8194" for port number.

If the application is not able to establish a connection to the local BBCOMM the call to `session.start()` will fail and return false. If the connection to the emapisvc service fails, OpenService call will return false.

3.13.1 EMSX API & Correlation ID

The `CorrelationID` ties the subscriptions and request response messages. The user would have to inspect the result to identify the source of the data and handle the message or the errors. Using the CorrelationID the user can immediately tell if it is emapisvc or mktdata. The `CorrelationID` is unique to the subscription only and not to the orders and routes.

The `CorrelationID` s are set when you send the request or submit the subscription. The `CorrelationID` s belong to the message. When an event fires, which is passed to the handler, this opens up the event and iterate through the message(s). There can be more than one message per event. Each message (MessageDataType) has a `.correlationID` property. The `CorrelationID` (CorrelationID datatype) is specified to a value and once the user submit it with the request in `sendRequest` call or when the user adds it to the individual subscription in the subscriptions list prior to the `session.subscribe` call.

3.14 Description of Request/Response Service

The request/response service can be used for both buy-side `EMSX<GO>` workflows or sell-side EMSX to EMSX (E2E) workflows.

The buy-side EMSX request/response supports all the basic buy-side execution management control via request/response service where the sell-side EMSX request/response supports additional sell-side workflow for acknowledging or rejecting an order coming in via E2E workflow.

3.14.1 Buy-Side Request/Response Service

EMSX API supports the following buy-side Request/Response services:- *Please note, the descriptions to the legacy request/response services are omitted from the description section.*

Request Name	Action
AssignTrader	Assign an order to another UUID.
CancelRouteEx	Cancel outstanding routes (placements).
CreateOrder	Create an order or stage an order into EMSX<GO>.
CreateOrderAndRouteEx	Create a new order and route in a single request.
CreateOrderAndRouteManually	Create the order and notify EMSX this is routed.
DeleteOrder	Delete an existing order in EMSX<GO>.
GetAllFieldMetaData	Get all field meta data in a response message.
GetBrokerStrategiesWithAssetClass	Get all broker strategy information and asset class data.
GetBrokerStrategyInfoWithAssetClass	Get all broker strategy info and asset class data.
GetBrokerWithAssetClass	Get all broker data with asset class in a response message.
GetFieldMetaData	Get field meta data in a reponse message.
GetTeams	Get team data in a response message.
GroupRouteEx	Submit the entire list as a single route to a basket algorithm.
ModifyOrder	Modify parent order.
ModifyRouteEx	Modify child route.
RouteEx	Route existing order.
RouteManuallyEx	Route manually and notify EMSX that it is routed.

Note: CreateOrderAndRouteEx can be used for both strategy and non-strategy broker destinations.

CreateOrderAndRouteManually is generally used for phone orders to brokers, where the actual placement is outside of EMSX<GO>.

RouteEx can be used for both strategy and non-strategy broker destinations.

RouteManuallyEx is generally used for phone orders to manually enter back the execution to EMSX<GO>.

3.14.2 Sell-Side Request/Response Service

EMSX API supports the following sell-side Request/Response services:- *Please note, the descriptions to the legacy request/response services are omitted from the description section.*

Request Name	Action
ManualFill	Request to manually fill a child route.
SellSideAck	Request to acknowlede an order on EMSX to EMSX setting.
SellSideReject	Request to reject an order on EMSX to EMSX setting.

Note: SellSideAck is used for EMSX to EMSX or E2E settings where sell-side EMSX<GO> is used to receive order from buy-side EMSX.

SellSideReject is used for EMSX to EMSX or E2E settings where sell-side EMSX<GO> is used to receive order from buy-side EMSX.

3.14.3 CFD & Odd Lot Flag

This is a feature that indicates CFD orders or to flag an odd lot in EMSX API. EMSX_CFD_FLAG is used to flag a particular order as CFD

- 0 = not flagged
- 1 = flagged

EMSX_ODD_LOT_FLAG is an odd lot is a quantity of stock that is less than 100 shares. A deal involving 100 shares or more is considered a round-lot transactions.

- 0 = not an odd lot / it won't fill odd lots
- 1 = odd lot

3.14.4 Date & Time Format

All date format except EMSX_QUEUED_TIME are in yyyymmdd format. All time format except EMSX_STRATEGY_END_TIME, EMSX_STRATEGY_START_TIME, and EMSX_RELEASE_TIME are in number of seconds from midnight.

The Strategy time zone is set using the EMSX<GO> function in the Bloomberg terminal under Routing Defaults section inside the Settings menu. In the Routing Defaults, the user can select Exchange vs. User time zone for strategy time zone. The default is the Exchange time.

Element	Description
EMSX_DATE	yyyymmdd
EMSX_GTD_DATE	yyyymmdd
EMSX_LAST_FILL_DATE	yyyymmdd
EMSX_QUEUED_DATE	yyyymmdd
EMSX_ROUTE_CREATE_DATE	yyyymmdd
EMSX_SETTLE_DATE	yyyymmdd
EMSX_QUEUED_TIME	hhmm
EMSX_RELEASE_TIME	hhmm (For the API, it is defaulted to the exchange time.)
EMSX_STRATEGY_END_TIME	hhmmss
EMSX_STRATEGY_START_TIME	hhmmss
EMSX_LAST_FILL_TIME	Number of seconds from midnight
EMSX_ROUTE_CREATE_TIME	Number of seconds from midnight
EMSX_ROUTE_LAST_UPDATE_TIME	Number of seconds from midnight
EMSX_TIME_STAMP	Number of seconds from midnight

The //blp/emsx.history and //blp/emsx.history.uat are set in date time objects unlike the //blp/emapisvc or //blp/emapisvc_beta.

3.14.5 Custom Notes & Free Text Fields

The EMSX API provides several different EMSX options for entering and using free text fields. Some of these free text fields can be used for an internal only workflow where the others can be used to communicate with the various execution counterparts.

The following elements are available on order and/or route subscription services. These elements will be passed to the external trading counterparts.

Element	Description
EMSX_ACCOUNT	29-character free text field (29+1 check digit), FIX Tag 1
EMSX_BASKET_NAME	20-character free text field (20+1 check digit)
EMSX_INVESTOR_ID	12-character free text field mostly used to identify Investor ID
EMSX_NOTES	43-character free text field (43+1 check digit), FIX Tag 58
EMSX_ORDER_REF_ID	15-character field (15+1 check digit) *order subscription only, not available for AIM users*
EMSX_ROUTE_REF_ID	15-character field (15+1 check digit) *route subscription only, not available for AIM users*
EMSX_TRADER_NOTES	43-character free text field (43+1 check digit), **internal & read only from API**

The following elements are available only for internal fields unless custom mapped to a custom FIX tag to a particular trading counterparty.

Warning: The following EMSX_CUSTOM_NOTE* elements are only available on order subscription service.

Element	Description
EMSX_CUSTOM_NOTE1	79-character free text field (79+1 check digit)
EMSX_CUSTOM_NOTE2	79-character free text field (79+1 check digit)
EMSX_CUSTOM_NOTE3	79-character free text field (79+1 check digit)
EMSX_CUSTOM_NOTE4	79-character free text field (79+1 check digit)
EMSX_CUSTOM_NOTE5	79-character free text field (79+1 check digit)

3.15 Buy-Side Request/Response Service

The EMSX API allows developers to use the Request/Response services for order and route creation, modification, queries related to orders and routes as well as EMSX Team details. Depending on the type of action required, the application programmer must create a specific request, populate it with required parameters and send that request to the EMSX API service, which provides the response. Communication with the request/response service requires the following steps:

1. Create a session (if session does not yet exist).
2. Connect session to //blp/emapisvc_beta or //blp/emapisvc service and start it.
3. Fetch a service object from the session representing emapisvc.
4. Use the service object from above to create a Request object of the desired type
5. Send request object via sendRequest method of session object, pass object of type EventQueue to the sendRequest.
6. Loop through the EventQueue object until event of type Event::RESPONSE is read.

These are initialized in the constructor as below and are then available for the life of the application for submission of various requests.

3.15.1 Assign Trader Request

The AssignTrader request allows EMSX API to reassign order to another user UUID. A typical setup will have the different UUID as another part of the TEAM setup for the order creater UUID. This will allow systematically

generated trades to be reassigned to another human trader if need be from the EMSX API.

Assigned trader must be in same EMBR<GO> group for this to work. EMBR<GO> is an internal Bloomberg function the account managers will use to set this feature on behalf of the client. The EMSX account manager will check off the ability to reassign before the AssignTrader request will work. Once this feature is on, trading on behalf other UUID feature will no longer work for that team.

Full code sample:-

Assign Trader cpp	Assign Trader cs	Assign Trader vba
Assign Trader java	Assign Trader py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("AssignTrader")

            request.append("EMSX_SEQUENCE", 3744303)
            request.append("EMSX_SEQUENCE", 3744341)

            request.set("EMSX_ASSIGNEE_TRADER_UUID", 12109783)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Output:- Without proper EMBR<GO> permssion.

```
C:\Users\tckim\OneDrive\_scripts>py -3 AssignTrader.py
Bloomberg - EMSX API Example - AssignTrader
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
```

(continues on next page)

(continued from previous page)

```
Request: AssignTrader = {
    EMSX_SEQUENCE[] = {
        4733955
    }
    EMSX_ASSIGNEE_TRADER_UUID = 7569479
}

Processing RESPONSE event
MESSAGE: ErrorInfo = {
    ERROR_CODE = 96233
    ERROR_MESSAGE = "Not Authorized"
}

CORRELATION ID: 3
MESSAGE TYPE: ErrorInfo
ERROR CODE: 96233       ERROR MESSAGE: Not Authorized
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.2 Broker Spec Request

The `BrokerSpec` request allows EMSX API users to call all the production broker strategy name and fields and FIX tags associated with the broker strategies. Unfortunately, this is currently only available for production broker strategy fields. The service name is `\\blp\emsx.brokerspec`.

Full code sample:-

Broker Spec cpp	Broker Spec cs	Broker Spec vba
Broker Spec java	Broker Spec py	

Call `//blp/emsx.brokerspec` service:-

```
SESSION_STARTED         = blpapi.Name("SessionStarted")
SESSION_STARTUP_FAILURE = blpapi.Name("SessionStartupFailure")
SERVICE_OPENED          = blpapi.Name("ServiceOpened")
SERVICE_OPEN_FAILURE    = blpapi.Name("ServiceOpenFailure")
ERROR_INFO              = blpapi.Name("ErrorInfo")
BROKER_SPEC             = blpapi.Name("BrokerSpec")

d_service="//blp/emsx.brokerspec" # The BrokerSpec service is only available in the␣
↪production environment
d_host="localhost"
d_port=8194
bEnd=False
```

Specify the UUID:-

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetBrokerSpecForUuid")

            request.set("uuid", 8049857)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Get broker code, strategy name, and strategy parameters

```
    brokers=msg.getElement("brokers")

num = brokers.numValues()

print "Number of Brokers: %d\n" % (num)

for broker in brokers.values():
    code = broker.getElement("code").getValue()
    assetClass = broker.getElement("assetClass").getValue()

    if broker.hasElement("strategyFixTag"):
        tag = broker.getElement("strategyFixTag").getValue()
        print "\nBroker code: %s\tclass: %s\ttag: %s" % (code,assetClass,tag)
        strats = broker.getElement("strategies")
        numStrats = strats.numValues()
        print"\tNo. of Strategies: %d" % (numStrats)
        for strat in strats.values():
            name = strat.getElement("name").getValue()
            fixVal = strat.getElement("fixValue").getValue()
            print "\n\tStrategy Name: %s\tFix Value: %s" % (name,fixVal)

            parameters = strat.getElement("parameters")

            numParams = parameters.numValues()

            print "\t\tNo. of Parameters: %d\n" % (numParams)

            for param in parameters.values():
                pname = param.getElement("name").getValue()
                tag = param.getElement("fixTag").getValue()
                required = param.getElement("isRequired").getValue()
                replaceable = param.getElement("isReplaceable").getValue()
```

(continues on next page)

(continued from previous page)

```
                print "\t\tParameter: %s\tTag: %d\tRequired: %s\tReplaceable: %s" %␣
↪(pname,tag,required,replaceable)

                typeName = param.getElement("type").getElement(0).name()

                vals = ""

                if typeName=="enumeration":

                    enumerators = param.getElement("type").getElement(0).getElement(
↪"enumerators")

                    for enum in enumerators.values():
                        vals = vals + enum.getElement("name").getValue() + "[" + enum.
↪getElement("fixValue").getValue() + "],"

                    if len(vals) > 0: vals = vals[:-1]


                elif typeName=="range":
                    rng = param.getElement("type").getElement(0)
                    mn = rng.getElement("min").getValue()
                    mx = rng.getElement("max").getValue()
                    st = rng.getElement("step").getValue()
                    vals = "min:%d max:%d step:%d" % (mn,mx,st)


                elif typeName=="string":
                    possVals = param.getElement("type").getElement(0).getElement(
↪"possibleValues")


                    for val in possVals.values():
                        vals = vals + val +","

                    if len(vals) > 0: vals = vals[:-1]


                if len(vals) > 0:
                    print "\t\t\tType: %s (%s)" % (typeName, vals)
                else:
                    print "\t\t\tType: %s" % (typeName)

    else:
        print "\nBroker code: %s\tclass: %s" % (code,assetClass)
        print"\tNo strategies\n"
```

Output:-

```
C:\Users\_scripts>py -3 BrokerSpec.py
Bloomberg - EMSX API Example - BrokerSpec
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}
```

(continues on next page)

(continued from previous page)

```
Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetBrokerSpecForUuid = {
    uuid = 6767714
}

Processing RESPONSE event
MESSAGE TYPE: BrokerSpec
Number of Brokers: 20


Broker code: BB class: Equity   tag: 9002
        No. of Strategies: 10

        Strategy Name: NONE     Fix Value: NONE
                No. of Parameters: 0


        Strategy Name: VWAP     Fix Value: VWAP
                No. of Parameters: 0


        Strategy Name: PARTICIPATE      Fix Value: PART
                No. of Parameters: 0


        Strategy Name: INLINE   Fix Value: INLINE
                No. of Parameters: 0


        Strategy Name: BIPS     Fix Value: 2
                No. of Parameters: 0


        Strategy Name: EP_PE    Fix Value: EP
                No. of Parameters: 0


        Strategy Name: PAIRS STRATEGY   Fix Value: PAIR
                No. of Parameters: 0


        Strategy Name: BEST EX  Fix Value: BEST-EX
                No. of Parameters: 0


        Strategy Name: ratest   Fix Value: ratest
                No. of Parameters: 0

        Time In Force:
                Name: DAY       Fix Value: 0
                Name: FOK       Fix Value: 4
                Name: GTC       Fix Value: 1
                Name: GTD       Fix Value: 6
```

(continues on next page)

(continued from previous page)

```
        Order Types:
                Name: LMT       Fix Value: 2
                Name: MKT       Fix Value: 1
                Name: SL        Fix Value: 4
                Name: ST        Fix Value: 3

        Handling Instructions:
                Name: ANY       Fix Value: 2
                Name: Auto      Fix Value: 1
                Name: DMA       Fix Value: 4
                Name: MAN       Fix Value: 3
                Name: ORD       Fix Value: 0

Broker code: BB class: Option
        No strategies

        Time In Force:
                Name: DAY       Fix Value: 0

        Order Types:
                Name: LMT       Fix Value: 2
                Name: MKT       Fix Value: 1

        Handling Instructions:
                Name: ANY       Fix Value: 2
                Name: AUTO      Fix Value: 1
                Name: MAN       Fix Value: 3

Broker code: EFIX       class: Equity   tag: 6005
        No. of Strategies: 53

        Strategy Name: TSTRIKE1 Fix Value: 2
                No. of Parameters: 0


        Strategy Name: INLINE   Fix Value: INLINE
                No. of Parameters: 0


        Strategy Name: STRATEGY8        Fix Value: 8
                No. of Parameters: 0


        Strategy Name: STRATEGY9        Fix Value: 9
                No. of Parameters: 0


        Strategy Name: STRATEGY10       Fix Value: 10
                No. of Parameters: 0


        Strategy Name: STRATEGY11       Fix Value: 11
                No. of Parameters: 0


        Strategy Name: STRATEGY12       Fix Value: 12
```

(continues on next page)

(continued from previous page)

```
            No. of Parameters: 0


        Strategy Name: STRATEGY13       Fix Value: 13
            No. of Parameters: 0


        Strategy Name: STRATEGY14       Fix Value: 14
            No. of Parameters: 0


        Strategy Name: STRATEGY15       Fix Value: 15
            No. of Parameters: 0


        Strategy Name: STRATEGY16       Fix Value: 16
            No. of Parameters: 0


        Strategy Name: STRATEGY17       Fix Value: 17
            No. of Parameters: 0


        Strategy Name: STRATEGY18       Fix Value: 18
            No. of Parameters: 0


        Strategy Name: STRATEGY19       Fix Value: 19
            No. of Parameters: 0


        Strategy Name: STRATEGY20       Fix Value: 20
            No. of Parameters: 0


        Strategy Name: STRATEGY21       Fix Value: 21
            No. of Parameters: 0


        Strategy Name: STRATEGY22       Fix Value: 22
            No. of Parameters: 0


        Strategy Name: STRATEGY23       Fix Value: 23
            No. of Parameters: 0


        Strategy Name: STRATEGY24       Fix Value: 24
            No. of Parameters: 0


        Strategy Name: STRATEGY25       Fix Value: 25
            No. of Parameters: 0


        Strategy Name: Merge    Fix Value: Merge
            No. of Parameters: 0
```

(continues on next page)

(continued from previous page)

```
Strategy Name: VWAP     Fix Value: GVW3
        No. of Parameters: 0


Strategy Name: TWAP     Fix Value: GTW3
        No. of Parameters: 0


Strategy Name: VP       Fix Value: GVP3
        No. of Parameters: 0


Strategy Name: VWAP2    Fix Value: 3
        No. of Parameters: 0


Strategy Name: ABC      Fix Value: 4
        No. of Parameters: 0


Strategy Name: TIME TEST        Fix Value: 1
        No. of Parameters: 0


Strategy Name: TIME TEST1       Fix Value: 40
        No. of Parameters: 0


Strategy Name: strategy 29      Fix Value: L
        No. of Parameters: 0


Strategy Name: strategy 30      Fix Value: 30
        No. of Parameters: 0


Strategy Name: ALGOT    Fix Value: TT
        No. of Parameters: 0


Strategy Name: Mike Sat Morning Fix Value: M3
        No. of Parameters: 0


Strategy Name: janurary Fix Value: jan
        No. of Parameters: 0


Strategy Name: test33           Fix Value: 10114
        No. of Parameters: 0


Strategy Name: iceberg  Fix Value: iceberg
        No. of Parameters: 0
```

(continues on next page)

(continued from previous page)

```
Strategy Name: Merge2   Fix Value: Merge2
        No. of Parameters: 0


Strategy Name: testwf   Fix Value: testwf
        No. of Parameters: 0


Strategy Name: TS Strike        Fix Value: y
        No. of Parameters: 0


Strategy Name: TS Strike        Fix Value: y
        No. of Parameters: 0


Strategy Name: strategy 30      Fix Value: 30
        No. of Parameters: 0


Strategy Name: Strategy 30      Fix Value: 30
        No. of Parameters: 0


Strategy Name: INLIN    Fix Value: INLINE
        No. of Parameters: 0


Strategy Name: TS Strike        Fix Value: y
        No. of Parameters: 0


Strategy Name: Strategy 30      Fix Value: 30
        No. of Parameters: 0


Strategy Name: SMART    Fix Value: SMART
        No. of Parameters: 0


Strategy Name: y029test Fix Value: 1029
        No. of Parameters: 0


Strategy Name: ra_test  Fix Value: ratest
        No. of Parameters: 0


Strategy Name: DEMO     Fix Value: D
        No. of Parameters: 0


Strategy Name: A        Fix Value: 2
        No. of Parameters: 0
```

(continues on next page)

(continued from previous page)

```
        Strategy Name: TEST1    Fix Value: T1
                No. of Parameters: 0


        Strategy Name: TEST2    Fix Value: T2
                No. of Parameters: 0


        Strategy Name: TEST3    Fix Value: T3
                No. of Parameters: 0


        Strategy Name: jeff     Fix Value: jeff
                No. of Parameters: 0

        Time In Force:
                Name: CLO       Fix Value: 7
                Name: DAY       Fix Value: 0
                Name: FOK       Fix Value: 4
                Name: GTC       Fix Value: 1
                Name: GTD       Fix Value: 6
                Name: GTX       Fix Value: 5
                Name: IOC       Fix Value: 3
                Name: OPG       Fix Value: A

        Order Types:
                Name: CD        Fix Value: Q
                Name: COVR      Fix Value: F
                Name: FUN       Fix Value: I
                Name: JP        Fix Value: N
                Name: LMT       Fix Value: 2
                Name: LOB       Fix Value: R
                Name: LOC       Fix Value: B
                Name: LOO       Fix Value: 6
                Name: MKT       Fix Value: 1
                Name: MOC       Fix Value: 5
                Name: MOO       Fix Value: X
                Name: OC        Fix Value: A
                Name: PEGG      Fix Value: P
                Name: RED       Fix Value: E
                Name: SL        Fix Value: 4
                Name: ST        Fix Value: 3

        Handling Instructions:
                Name: ANY       Fix Value: 2
                Name: AUTO      Fix Value: 1
                Name: MAN       Fix Value: 3

Broker code: EFIX       class: Future   tag: 1000
        No. of Strategies: 6

        Strategy Name: test 2   Fix Value: 200
                No. of Parameters: 0


        Strategy Name: test     Fix Value: 100
                No. of Parameters: 0
```

(continues on next page)

(continued from previous page)

```
        Strategy Name: time test        Fix Value: time
                No. of Parameters: 0


        Strategy Name: Range test       Fix Value: rng
                No. of Parameters: 0


        Strategy Name: test3    Fix Value: I
                No. of Parameters: 0


        Strategy Name: DEMO     Fix Value: D
                No. of Parameters: 0

        Time In Force:
                Name: DAY       Fix Value: 0
                Name: GTC       Fix Value: 1
                Name: GTD       Fix Value: 6
                Name: GTI       Fix Value: 8
                Name: GTT       Fix Value: 9
                Name: IOC       Fix Value: 3

        Order Types:
                Name: LMT       Fix Value: 2
                Name: MKT       Fix Value: 1
                Name: MOC       Fix Value: 5
                Name: SL        Fix Value: 4
                Name: ST        Fix Value: 3

        Handling Instructions:
                Name: ANY       Fix Value: 2
                Name: AUTO      Fix Value: 1
                Name: DOT       Fix Value: 4
                Name: MAN       Fix Value: 3
...
```

3.15.3 Cancel Order Extended Request

In `EMSX<GO>` there is a feature that allows the user to cancel the parent order and child routes associated with the parent order in a single call. The `CancelOrderEx` request replicates this `EMSX<GO>` UI feature.

However, unlike the CancelRouteEx request which changes the parent order state into `Assigned`, this request will permanently place the order in an inoperable `Cancel` state.

Important: Please note this request does not work for AIM users. This request only works for standalone EMSX API user.

Full code sample:-

Cancel Order cs	Cancel Order py	

Hint: Please right click on the top code sample link to open in a new tab.

3.15.4 Cancel Route Extended Request

In EMSX<GO> we have a notion of parent order and child routes. The CancelRoute request is to effectively send out a cancellation request to the execution venue of the current live route. Submission of CancelRoute does not automatically cancel the outstanding route. This action needs to be acknowledged and performed by the execution venue of the route.

Full code sample:-

Cancel Route cpp	Cancel Route cs	Cancel Route vba
Cancel Route java	Cancel Route py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("CancelRoute")

            #request.set("EMSX_REQUEST_SEQ", 1)
            #request.set("EMSX_TRADER_UUID", 1234567)        # UUID of trader who owns the order

            routes = request.getElement("ROUTES")

            route = routes.appendElement()
            route.getElement("EMSX_SEQUENCE").setValue(3744354)
            route.getElement("EMSX_ROUTE_ID").setValue(1)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\tckim\OneDrive\_scripts>py -3 CancelOrderEx.py
Bloomberg - EMSX API Example - CancelOrderEx
```

(continues on next page)

(continued from previous page)

```
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: CancelOrderEx = {
    EMSX_SEQUENCE[] = {
        4733955
    }
}

Processing RESPONSE event
MESSAGE: CancelOrderEx = {
    STATUS = 1
    MESSAGE = "Order cancellation request sent to broker"
}

CORRELATION ID: 3
MESSAGE TYPE: CancelOrderEx
STATUS: 1       MESSAGE: Order cancellation request sent to broker
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.5 Create Basket Request

Creating a basket requires the user to create a request from the service object of type `CreateBasket` and fill in the required fields before submitting the request.

The `CreateBasket` request creates a basket with the list of securities. This maintains a list or a basket from a portfolio perspective.

Currently, in EMSX API this is a two-step process.

The first step is for the user to use `CreateOrder` request to create the orders and capture the `EMSX_SEQUENCE` from the response message.

The second step is to include the `EMSX_SEQUENCE` number inside an array to add the orders into a basket and use the `EMSX_BASKET_NAME` element in the `CreateBasket` request to specify the name of the basket.

Full code sample:-

Create Basket cpp	Create Basket cs	Create Basket vba
Create Basket java	Create Basket py	

Hint: Please right click on the top code sample link to open in a new tab.

```
  def processServiceStatusEvent(self,event,session):
     print("Processing SERVICE_STATUS event")

     for msg in event:

         if msg.messageType() == SERVICE_OPENED:
             print("Service opened...")

             service = session.getService(d_service)

             request = service.createRequest("CreateBasket")

             # define the basket name
             request.set("EMSX_BASKET_NAME", "TestBasket")

             # add any number of orders
             request.append("EMSX_SEQUENCE", 4313227)
             request.append("EMSX_SEQUENCE", 4313228)
             #request.append("EMSX_SEQUENCE", 4313184)

             print("Request: %s" % request.toString())

             self.requestID = blpapi.CorrelationId()

             session.sendRequest(request, correlationId=self.requestID )

         elif msg.messageType() == SERVICE_OPEN_FAILURE:
             print("Error: Service failed to open")
```

Output:-

```
C:\Users\_scripts>py -3 CreateBasket.py
Bloomberg - EMSX API Example - CreateBasket
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: CreateBasket = {
    EMSX_BASKET_NAME = "TestBasket"
    EMSX_SEQUENCE[] = {
        4733961, 4733962
    }
}

Processing RESPONSE event
MESSAGE: CreateBasket = {
    EMSX_SEQUENCE[] = {
```

(continues on next page)

(continued from previous page)

```
        4733961, 4733962
    }
    MESSAGE = "Orders added to Basket"
}

CORRELATION ID: 3
MESSAGE TYPE: CreateBasket
EMSX_SEQUENCE: 4733961  MESSAGE: Orders added to Basket
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.6 Create Order Request

Creating an order requires the user to create a request from the service object of type `CreateOrder` and fill in the required fields before submitting the request.

If the handling instruction is for DMA access or any other non-standard handling instructions, EMSX API will not allow users to stage the order from the EMSX API unless the broker enables the broker code for EMSX API. This is also true for custom Time in Force fields. Any non-standard TIF will also be restricted from staging unless the broker enables the broker code for EMSX API.

Full code sample:-

Create Order cpp	Create Order cs	Create Order vba
Create Order java	Create Order py	

Hint: Please right click on the top code sample link to open in a new tab.

```
1  def processServiceStatusEvent(self,event,session):
2      print "Processing SERVICE_STATUS event"
3
4      for msg in event:
5
6          if msg.messageType() == SERVICE_OPENED:
7              print "Service opened..."
8
9              service = session.getService(d_service)
10
11             request = service.createRequest("CreateOrder")
12
13             # The fields below are mandatory
14             request.set("EMSX_TICKER", "IBM US Equity")
15             request.set("EMSX_AMOUNT", 1000)
16             request.set("EMSX_ORDER_TYPE", "MKT")
17             request.set("EMSX_TIF", "DAY")
18             request.set("EMSX_HAND_INSTRUCTION", "ANY")
```

(continues on next page)

(continued from previous page)

```
            request.set("EMSX_SIDE", "BUY")

            # The fields below are optional
            #request.set("EMSX_ACCOUNT","TestAccount")
            #request.set("EMSX_BASKET_NAME", "HedgingBasket")
            #request.set("EMSX_BROKER", "BMTB")
            #request.set("EMSX_CFD_FLAG", "1")
            #request.set("EMSX_CLEARING_ACCOUNT", "ClrAccName")
            #request.set("EMSX_CLEARING_FIRM", "FirmName")
            #request.set("EMSX_CUSTOM_NOTE1", "Note1")
            #request.set("EMSX_CUSTOM_NOTE2", "Note2")
            #request.set("EMSX_CUSTOM_NOTE3", "Note3")
            #request.set("EMSX_CUSTOM_NOTE4", "Note4")
            #request.set("EMSX_CUSTOM_NOTE5", "Note5")
            #request.set("EMSX_EXCHANGE_DESTINATION", "ExchDest")
            #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
            #request.set("EMSX_GET_WARNINGS", "0")
            #request.set("EMSX_GTD_DATE", "20170105")
            #request.set("EMSX_INVESTOR_ID", "InvID")
            #request.set("EMSX_LIMIT_PRICE", 123.45)
            #request.set("EMSX_LOCATE_BROKER", "BMTB")
            #request.set("EMSX_LOCATE_ID", "SomeID")
            #request.set("EMSX_LOCATE_REQ", "Y")
            #request.set("EMSX_NOTES", "Some notes")
            #request.set("EMSX_ODD_LOT", "0")
            #request.set("EMSX_ORDER_ORIGIN", "")
            #request.set("EMSX_ORDER_REF_ID", "UniqueID")
            #request.set("EMSX_P_A", "P")
            #request.set("EMSX_RELEASE_TIME", 1259)
            #request.set("EMSX_REQUEST_SEQ", 1001)
            #request.set("EMSX_SETTLE_CURRENCY", "USD")
            #request.set("EMSX_SETTLE_DATE", 20170106)
            #request.set("EMSX_SETTLE_TYPE", "T+2")
            #request.set("EMSX_STOP_PRICE", 123.5)

            print "Request: %s" % request.toString()

        self.requestID = blpapi.CorrelationId()

        session.sendRequest(request, correlationId=self.requestID )

    elif msg.messageType() == SERVICE_OPEN_FAILURE:
        print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 CreateOrder.py
Bloomberg - EMSX API Example - CreateOrder
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
```

(continues on next page)

(continued from previous page)

```
Processing SERVICE_STATUS event
Service opened...
Request: CreateOrder = {
    EMSX_TICKER = "MSFT US Equity"
    EMSX_AMOUNT = 1100
    EMSX_ORDER_TYPE = MKT
    EMSX_TIF = DAY
    EMSX_HAND_INSTRUCTION = "ANY"
    EMSX_SIDE = BUY
}

Processing RESPONSE event
MESSAGE: CreateOrder = {
    EMSX_SEQUENCE = 4733955
    MESSAGE = "Order created"
}

CORRELATION ID: 3
MESSAGE TYPE: CreateOrder
EMSX_SEQUENCE: 4733955  MESSAGE: Order created
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.7 Create Order and Route Extended Request

The `CreateOrderAndRouteEx` request can be used for both strategy and non-strategy broker destinations. Creating an order and routing with strategy requires the user to create a request from the service object of type ‘‘ CreateOrderAndRouteEx‘‘ and fill in the required fields before submitting the request.

Note:

> The user will first need to use various `Get____` requests to obtain all the necessary information to use the broker strategies the user is enabled for, returned in response. Subsequently, the user can then request `GetBrokerStrategiesWithAssetClass` to get all the broker strategies user is enabled for that particular broker code and asset class.

Lastly, `GetBrokerStrategyInfoWithAssetClass` will get all the fields for the provided broker strategy in the particular order in which they need to be submitted in `CreateOrderAndRouteEx` and `RouteEx` requests.

Full code sample:-

Create Order And Route Extended cpp	Create Order And Route Extended cs	Create Order And Route Extended vba
Create Order And Route Extended java	Create Order And Route Extended py	

Hint: Please right click on the top code sample link to open in a new tab.

```
        def processServiceStatusEvent(self,event,session):
            print "Processing SERVICE_STATUS event"

            for msg in event:

                if msg.messageType() == SERVICE_OPENED:
                    print "Service opened..."

                    service = session.getService(d_service)

                    request = service.createRequest("CreateOrderAndRouteEx")

                    # The fields below are mandatory
                    request.set("EMSX_TICKER", "IBM US Equity")
                    request.set("EMSX_AMOUNT", 1000)
                    request.set("EMSX_ORDER_TYPE", "MKT")
                    request.set("EMSX_TIF", "DAY")
                    request.set("EMSX_HAND_INSTRUCTION", "ANY")
                    request.set("EMSX_SIDE", "BUY")
                    request.set("EMSX_BROKER", "BB")

                    # The fields below are optional
                    #request.set("EMSX_ACCOUNT","TestAccount")
```

Output:-

```
C:\Users\_scripts>py -3 CreateOrderAndRouteEx.py
Bloomberg - EMSX API Example - CreateOrderAndRouteEx
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: CreateOrderAndRouteEx = {
    EMSX_TICKER = "FB US Equity"
    EMSX_AMOUNT = 1000
    EMSX_ORDER_TYPE = MKT
    EMSX_TIF = DAY
    EMSX_HAND_INSTRUCTION = "ANY"
    EMSX_SIDE = SELL
    EMSX_BROKER = "BMTB"
    EMSX_ACCOUNT = "testAccount"
    EMSX_NOTES = "blah blah blah"
    EMSX_ORDER_REF_ID = "UniqueID"
    EMSX_P_A = "A"
    EMSX_ROUTE_REF_ID = "UniqueID2"
    EMSX_STRATEGY_PARAMS = {
        EMSX_STRATEGY_NAME = "VWAP"
```

(continues on next page)

(continued from previous page)

```
        EMSX_STRATEGY_FIELD_INDICATORS[] = {
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 0
            }
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 0
            }
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 1
            }
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 1
            }
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 1
            }
            EMSX_STRATEGY_FIELD_INDICATORS = {
                EMSX_FIELD_INDICATOR = 1
            }
        }
        EMSX_STRATEGY_FIELDS[] = {
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = "09:30:00"
            }
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = "10:30:00"
            }
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = ""
            }
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = ""
            }
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = ""
            }
            EMSX_STRATEGY_FIELDS = {
                EMSX_FIELD_DATA = ""
            }
        }
    }
}

Processing RESPONSE event
MESSAGE: CreateOrderAndRouteEx = {
    EMSX_SEQUENCE = 4733965
    EMSX_ROUTE_ID = 1
    MESSAGE = "Order created and routed"
}

CORRELATION ID: 3
MESSAGE TYPE: CreateOrderAndRouteEx
EMSX_SEQUENCE: 4733965  EMSX_ROUTE_ID: 1        MESSAGE: Order created and routed
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}
```

(continues on next page)

(continued from previous page)

```
Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.8 Create Order And Route Manually Request

The `CreateOrderAndRouteManually` request is generally used for phone orders where the placement is external to EMSX API. This request creates an order and notifies EMSX<GO> that this order is routed to the execution venue.

Full code sample:-

Create Order And Route Manually cpp	Create Order And Route Manually cs	Create Order And Route Manually vba
Create Order And Route Manually java	Create Order And Route Manually py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("CreateOrderAndRouteManually")

            # The fields below are mandatory
            request.set("EMSX_TICKER", "IBM US Equity")
            request.set("EMSX_AMOUNT", 1000)
            request.set("EMSX_ORDER_TYPE", "MKT")
            request.set("EMSX_TIF", "DAY")
            request.set("EMSX_HAND_INSTRUCTION", "ANY")
            request.set("EMSX_SIDE", "BUY")
            request.set("EMSX_BROKER", "BB")

            # The fields below are optional
            #request.set("EMSX_ACCOUNT","TestAccount")
            #request.set("EMSX_CFD_FLAG", "1")
            #request.set("EMSX_CLEARING_ACCOUNT", "ClrAccName")
            #request.set("EMSX_CLEARING_FIRM", "FirmName")
            #request.set("EMSX_EXCHANGE_DESTINATION", "ExchDest")
            #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
            #request.set("EMSX_GET_WARNINGS", "0")
            #request.set("EMSX_GTD_DATE", "20170105")
            #request.set("EMSX_INVESTOR_ID", "InvID")
            #request.set("EMSX_LIMIT_PRICE", 123.45)
```

(continues on next page)

(continued from previous page)

```
            #request.set("EMSX_LOCATE_BROKER", "BMTB")
            #request.set("EMSX_LOCATE_ID", "SomeID")
            #request.set("EMSX_LOCATE_REQ", "Y")
            #request.set("EMSX_NOTES", "Some notes")
            #request.set("EMSX_ODD_LOT", "0")
            #request.set("EMSX_ORDER_ORIGIN", "")
            #request.set("EMSX_ORDER_REF_ID", "UniqueID")
            #request.set("EMSX_P_A", "P")
            #request.set("EMSX_RELEASE_TIME", 1259)
            #request.set("EMSX_REQUEST_SEQ", 1001)
            #request.set("EMSX_SETTLE_DATE", 20170106)
            #request.set("EMSX_STOP_PRICE", 123.5)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.9 Delete Order Request

The `DeleteOrder` request deletes an existing order in EMSX<GO>. This is not the same action as canceling the parent order. In fact, EMSX API does not expose Cancel Order status as in EMSX<GO>.

The primary reason behind this is because the cancel rrder in EMSX<GO> really just puts an order in an inoperable state and doesn't really serve any meaningful function.

Full code sample:-

Delete Order cpp	Delete Order cs	Delete Order vba
Delete Order java	Delete Order py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("DeleteOrder")

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.getElement("EMSX_SEQUENCE").appendValue(3744363)
```

(continues on next page)

(continued from previous page)

```
            request.getElement("EMSX_SEQUENCE").appendValue(3744364)


            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 DeleteOrder.py
Bloomberg - EMSX API Example - DeleteOrder
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: DeleteOrder = {
    EMSX_SEQUENCE[] = {
        4733961
    }
}

Processing RESPONSE event
MESSAGE: DeleteOrder = {
    STATUS = 0
    MESSAGE = "Order deleted"
}

CORRELATION ID: 3
MESSAGE TYPE: DeleteOrder
STATUS: 0       MESSAGE: Order deleted
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.10 Get All Field Metadata Request

The `GetAllFiedlMetaData` request provides all field metadata in a response message.

Full code sample:-

Get All Field Meta Data cpp	Get All Field Meta Data cs	Get All Field Meta Data vba
Get All Field Meta Data java	Get All Field Meta Data py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetAllFieldMetaData")

            #request.set("EMSX_REQUEST_SEQ", 1)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Process response messages:-

```
def processResponseEvent(self, event):
    print "Processing RESPONSE event"

    for msg in event:

        print "MESSAGE: %s" % msg.toString()
        print "CORRELATION ID: %d" % msg.correlationIds()[0].value()


        if msg.correlationIds()[0].value() == self.requestID.value():
            print "MESSAGE TYPE: %s" % msg.messageType()

            if msg.messageType() == ERROR_INFO:
                errorCode = msg.getElementAsInteger("ERROR_CODE")
                errorMessage = msg.getElementAsString("ERROR_MESSAGE")
                print "ERROR CODE: %d\tERROR MESSAGE: %s" % (errorCode,errorMessage)
            elif msg.messageType() == GET_ALL_FIELD_METADATA:

                md = msg.getElement("MetaData")

                for e in md.values():

                    emsx_field_name = e.getElementAsString("EMSX_FIELD_NAME")
                    emsx_disp_name = e.getElementAsString("EMSX_DISP_NAME")
                    emsx_type = e.getElementAsString("EMSX_TYPE")
```

(continues on next page)

(continued from previous page)

```
                    emsx_level = e.getElementAsInteger("EMSX_LEVEL")
                    emsx_len = e.getElementAsInteger("EMSX_LEN")

                    print "MetaData: %s,%s,%s,%d,%d" % (emsx_field_name, emsx_disp_
↪name, emsx_type, emsx_level, emsx_len)

            global bEnd
            bEnd = True

def processMiscEvents(self, event):

    print "Processing " + event.eventType() + " event"

    for msg in event:

        print "MESSAGE: %s" % (msg.tostring())
```

Output:-

```
C:\Users\_scripts>py -3 GetAllFieldMetaData.py
Bloomberg - EMSX API Example - GetAllFieldMetaData
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetAllFieldMetaData = {
}

Processing RESPONSE event
MESSAGE: GetAllFieldMetaData = {
    MetaData[] = {
        MetaData = {
            EMSX_FIELD_NAME = "MSG_TYPE"
            EMSX_DISP_NAME = "Msg Type"
            EMSX_TYPE = String
            EMSX_LEVEL = 0
            EMSX_LEN = 1
        }
        MetaData = {
            EMSX_FIELD_NAME = "MSG_SUB_TYPE"
            EMSX_DISP_NAME = "Msg Sub Type"
            EMSX_TYPE = String
            EMSX_LEVEL = 0
            EMSX_LEN = 1
        }
        MetaData = {
            EMSX_FIELD_NAME = "EVENT_STATUS"
            EMSX_DISP_NAME = "Msg Status"
            EMSX_TYPE = Int32
            EMSX_LEVEL = 0
```

(continues on next page)

(continued from previous page)

```
            EMSX_LEN = 10
        }
        MetaData = {
            EMSX_FIELD_NAME = "API_SEQ_NUM"
            EMSX_DISP_NAME = "Api Sequence"
            EMSX_TYPE = Int64
            EMSX_LEVEL = 0
            EMSX_LEN = 20
        }
        MetaData = {
            EMSX_FIELD_NAME = "EMSX_SEQUENCE"
            EMSX_DISP_NAME = "Sequence #"
            EMSX_TYPE = Int32
            EMSX_LEVEL = 27
            EMSX_LEN = 10
        }
        MetaData = {
            EMSX_FIELD_NAME = "EMSX_ROUTE_ID"
            EMSX_DISP_NAME = "Tran No"
            EMSX_TYPE = Int32
            EMSX_LEVEL = 11
            EMSX_LEN = 10
        }
        MetaData = {
            EMSX_FIELD_NAME = "EMSX_FILL_ID"
            EMSX_DISP_NAME = "Fill Id"
            EMSX_TYPE = Int32
            EMSX_LEVEL = 2
            EMSX_LEN = 10
        }
        MetaData = {
            EMSX_FIELD_NAME = "EMSX_SIDE"
            EMSX_DISP_NAME = "B/S"
            EMSX_TYPE = String
            EMSX_LEVEL = 17
            EMSX_LEN = 4
       ...

       ...
        MetaData = {
            EMSX_FIELD_NAME = "EMSX_LEG_FILL_TICKER"
            EMSX_DISP_NAME = "Leg Fill Ticker"
            EMSX_TYPE = String
            EMSX_LEVEL = 2
            EMSX_LEN = 32
        }
    }
}

CORRELATION ID: 3
MESSAGE TYPE: GetAllFieldMetaData
MetaData: MSG_TYPE,Msg Type,String,0,1
MetaData: MSG_SUB_TYPE,Msg Sub Type,String,0,1
MetaData: EVENT_STATUS,Msg Status,Int32,0,10
MetaData: API_SEQ_NUM,Api Sequence,Int64,0,20
MetaData: EMSX_SEQUENCE,Sequence #,Int32,27,10
...
```

(continues on next page)

(continued from previous page)

```
...
MetaData: EMSX_ROUTE_AS_OF_TIME_MICROSEC,Route As of Time,Time,2,20
MetaData: EMSX_AS_OF_DATE,Order/Route As of Date,Date,24,8
MetaData: EMSX_AS_OF_TIME_MICROSEC,Order/Route As of Time,Time,24,20
MetaData: EMSX_LEG_FILL_SIDE,Leg Fill Side,String,2,3
MetaData: EMSX_LEG_FILL_DATE_ADDED,Leg Fill Date Added,Date,2,8
MetaData: EMSX_LEG_FILL_TIME_ADDED,Leg fill Time Added,Time,2,20
MetaData: EMSX_LEG_FILL_SHARES,Leg Fill Shares,Double,2,15
MetaData: EMSX_LEG_FILL_PRICE,Leg Fill Price,Double,2,15
MetaData: EMSX_LEG_FILL_SEQ_NO,Leg Fill Seq No,Int32,2,10
MetaData: EMSX_LEG_FILL_TICKER,Leg Fill Ticker,String,2,32
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.11 Get Broker Strategies with Asset Class Request

The `GetBrokerStrategiesWithAssetClass` request provides all broker strategy fields with asset class data in a response message.

Full code sample:-

Get Broker Strategies With Asset Class cpp	Get Broker Strategies With Asset Class cs	Get Broker Strategies With Asset Class vba
Get Broker Strategies With Asset Class java	Get Broker Strategies With Asset Class py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetBrokerStrategiesWithAssetClass")

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.set("EMSX_ASSET_CLASS","EQTY")  # one of EQTY, OPT, FUT or␣
↪MULTILEG_OPT
            request.set("EMSX_BROKER","BMTB")

```

(continues on next page)

(continued from previous page)

```
            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 GetBrokerStrategiesWithAssetClass.py
Bloomberg - EMSX API Example - GetBrokerStrategiesWithAssetClass
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetBrokerStrategiesWithAssetClass = {
    EMSX_ASSET_CLASS = EQTY
    EMSX_BROKER = "PAIR"
}

Processing RESPONSE event
MESSAGE: GetBrokerStrategiesWithAssetClass = {
    EMSX_STRATEGIES[] = {
        ""
    }
}

CORRELATION ID: 3
MESSAGE TYPE: GetBrokerStrategiesWithAssetClass
EMSX_STRATEGY:
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.12 Get Broker Strategy Info with Asset Class Request

The `GetBrokerStrategyInfoWithAssetClass` request provides all broker strategy information fields with asset classdata in a response message.

Full code sample:-

Get Broker Strategy Info With Asset Class cpp	Get Broker Strategy Info With Asset Class cs	Get Broker Strategy Info With Asset Class vba
Get Broker Strategy Info With Asset Class java	Get Broker Strategy Info With Asset Class py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetBrokerStrategyInfoWithAssetClass")

            request.set("EMSX_REQUEST_SEQ", 1)

            request.set("EMSX_ASSET_CLASS","EQTY")  # one of EQTY, OPT, FUT or␣
↪MULTILEG_OPT
            request.set("EMSX_BROKER","BMTB")
            request.set("EMSX_STRATEGY","VWAP")

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 GetBrokerStrategyInfoWithAssetClass.py
Bloomberg - EMSX API Example - GetBrokerStrategyInfoWithAssetClass
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetBrokerStrategyInfoWithAssetClass = {
    EMSX_REQUEST_SEQ = 1
    EMSX_ASSET_CLASS = EQTY
    EMSX_BROKER = "BMTB"
```

(continues on next page)

(continued from previous page)

```
    EMSX_STRATEGY = "VWAP"
}

Processing RESPONSE event
MESSAGE: GetBrokerStrategyInfoWithAssetClass = {
    EMSX_STRATEGY_INFO[] = {
        EMSX_STRATEGY_INFO = {
            FieldName = "Start Time"
            Disable = 0
            StringValue = ""
        }
        EMSX_STRATEGY_INFO = {
            FieldName = "End Time"
            Disable = 0
            StringValue = ""
        }
        EMSX_STRATEGY_INFO = {
            FieldName = "Max % Volume"
            Disable = 0
            StringValue = ""
        }
        EMSX_STRATEGY_INFO = {
            FieldName = "Discretion"
            Disable = 0
            StringValue = ""
        }
        EMSX_STRATEGY_INFO = {
            FieldName = "Display Qty"
            Disable = 0
            StringValue = ""
        }
        EMSX_STRATEGY_INFO = {
            FieldName = "FltLmtType"
            Disable = 0
            StringValue = ""
        }
    }
}

CORRELATION ID: 3
MESSAGE TYPE: GetBrokerStrategyInfoWithAssetClass
EMSX_STRATEGY_INFO: Start Time, 0,
EMSX_STRATEGY_INFO: End Time, 0,
EMSX_STRATEGY_INFO: Max % Volume, 0,
EMSX_STRATEGY_INFO: Discretion, 0,
EMSX_STRATEGY_INFO: Display Qty, 0,
EMSX_STRATEGY_INFO: FltLmtType, 0,
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.13 Get Brokers with Asset Class Request

The `GetBrokersWithAssetClass` request provides all broker information with asset class data in a response message.

Full code sample:-

Get Brokers With Asset Class cpp	Get Brokers With Asset Class cs	Get Brokers With Asset Class vba
Get Brokers With Asset Class java	Get Brokers With Asset Class py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetBrokersWithAssetClass")

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.set("EMSX_ASSET_CLASS","EQTY")  # one of EQTY, OPT, FUT or␣
↪MULTILEG_OPT

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
                    print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 GetBrokersWithAssetClass.py
Bloomberg - EMSX API Example - GetBrokersWithAssetClass
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetBrokersWithAssetClass = {
    EMSX_ASSET_CLASS = EQTY
```

(continues on next page)

(continued from previous page)

```
}

Processing RESPONSE event
MESSAGE: GetBrokersWithAssetClass = {
    EMSX_BROKERS[] = {
        "API", "BB", "BEXE", "BMTB", "EEUE", "EFIX", "RFQ", "TKOR"
    }
}

CORRELATION ID: 3
MESSAGE TYPE: GetBrokersWithAssetClass
EMSX_BROKER: API
EMSX_BROKER: BB
EMSX_BROKER: BEXE
EMSX_BROKER: BMTB
EMSX_BROKER: EEUE
EMSX_BROKER: EFIX
EMSX_BROKER: RFQ
EMSX_BROKER: TKOR
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.14 Get Field Metadata Request

The `GetFieldMetaData` request provides all field metadata in a response message.

Full code sample:-

Get Field Meta Data cpp	Get Field Meta Data cs	Get Field Meta Data vba
Get Field Meta Data java	Get Field Meta Data py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetFieldMetaData")

            #request.set("EMSX_REQUEST_SEQ", 1)
```

(continues on next page)

(continued from previous page)

```
            request.getElement("EMSX_FIELD_NAMES").appendValue("EMSX_TICKER")
            request.getElement("EMSX_FIELD_NAMES").appendValue("EMSX_P_A")

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.15 Get Teams Request

The `GetTeams` request provides all the team details in a response message.

Full code sample:-

Get Teams cpp	Get Teams cs	Get Teams vba
Get Teams java	Get Teams py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetTeams")

            #request.set("EMSX_REQUEST_SEQ", 1)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.16 Get Trade Desks Request

The `GetTradeDesks` is AIM specific request and provides all the trade desk details in a response message.

Full code sample:-

Get Trade Desks cs	
Get Trade Desks py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetTradeDesks")

            #request.set("EMSX_REQUEST_SEQ", 1)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.17 Get Traders Request

The `GetTraders` is AIM specific request and provides all the traders details in a response message.

Full code sample:-

Get Traders cs	
Get Traders py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("GetTraders")
```

(continues on next page)

(continued from previous page)

```
            #request.set("EMSX_REQUEST_SEQ", 1)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.18 Group Route Extended Request

The `GroupRouteEx` request submits an entire list as a single route to a basket/program broker strategy destination.

This request should only be used if the intention is to submit an entire list or basket of securities to a single broker strategy destination. This should not be confused with maintaining a list or a basket from a portfolio perspective.

Currently, this is a three-step process in EMSX API.

The first step is for the user will need to use `CreateOrder` request to create the order. Once the orders are created, the user will use `CreateBasket` request to create the basket or list of orders and use `EMSX_BASKET_NAME` element to specify the basket name.

The next step is to submit the list using `GroupRouteEx` request and include the `EMSX_SEQUENCE` number inside the array.

Important: Please remember that the application does need to wait for confirmation of the basket creation to trigger the the `GroupRouteEx` request. The `GroupRouteEx` request is NOT independent of the basket creation for routing (placements).

Full code sample:-

Group Route Extended cpp	Group Route Extended cs	Group Route Extended vba
Group Route Extended java	Group Route Extended py	

Hint: Please right click on the top code sample link to open in a new tab.

```
        def processServiceStatusEvent(self,event,session):
            print "Processing SERVICE_STATUS event"

            for msg in event:

                if msg.messageType() == SERVICE_OPENED:
                    print "Service opened..."

                    service = session.getService(d_service)

                    request = service.createRequest("GroupRouteEx")

```

(continues on next page)

(continued from previous page)

```
                    # Multiple order numbers can be added
                    request.append("EMSX_SEQUENCE", 3745211)
                    request.append("EMSX_SEQUENCE", 3745212)
                    request.append("EMSX_SEQUENCE", 3745213)

                    # The fields below are mandatory
                    request.set("EMSX_AMOUNT_PERCENT", 100)  # Note the amount here
↪is %age of order amount
                    request.set("EMSX_BROKER", "BMTB");

                    # For GroupRoute, the below values need to be added, but are
↪taken
                    # from the original order when the route is created.
                    request.set("EMSX_HAND_INSTRUCTION", "ANY")
                    request.set("EMSX_ORDER_TYPE", "MKT")
                    request.set("EMSX_TICKER", "IBM US Equity")
                    request.set("EMSX_TIF", "DAY")

                    # The fields below are optional
                    #request.set("EMSX_ACCOUNT","TestAccount")
                    #request.set("EMSX_BOOKNAME","BookName")
                    #request.set("EMSX_CFD_FLAG", "1")
                    #request.set("EMSX_CLEARING_ACCOUNT", "ClrAccName")
                    #request.set("EMSX_CLEARING_FIRM", "FirmName")
                    #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
                    #request.set("EMSX_GET_WARNINGS", "0")
                    #request.set("EMSX_GTD_DATE", "20170105")
                    #request.set("EMSX_LIMIT_PRICE", 123.45)
                    #request.set("EMSX_LOCATE_BROKER", "BMTB")
                    #request.set("EMSX_LOCATE_ID", "SomeID")
                    #request.set("EMSX_LOCATE_REQ", "Y")
                    #request.set("EMSX_NOTES", "Some notes")
                    #request.set("EMSX_ODD_LOT", "0")
                    #request.set("EMSX_P_A", "P")
                    #request.set("EMSX_RELEASE_TIME", 1259)
                    #request.set("EMSX_REQUEST_SEQ", 1001)
                    #request.set("EMSX_STOP_PRICE", 123.5)
                    #request.set("EMSX_TRADER_UUID", 1234567)

                    # Set the Request Type if this is for multi-leg orders
                    # only valid for options
                    '''
                    requestType = request.getElement("EMSX_REQUEST_TYPE")
                    requestType.setChoice("Multileg")
                    multileg = requestType.getElement("Multileg")
                    multileg.setElement("EMSX_AMOUNT",10)
                    multileg.getElement("EMSX_ML_RATIO").appendValue(2)
                    multileg.getElement("EMSX_ML_RATIO").appendValue(3)
                    '''

                    # Add the Route Ref ID values
                    routeRefIDPairs = request.getElement("EMSX_ROUTE_REF_ID_PAIRS")
                    route1 = routeRefIDPairs.appendElement()
                    route1.setElement("EMSX_ROUTE_REF_ID","MyRouteRef1")
                    route1.setElement("EMSX_SEQUENCE",3745211)

                    route2 = routeRefIDPairs.appendElement();
```

(continues on next page)

(continued from previous page)

```
                    route2.setElement("EMSX_ROUTE_REF_ID","MyRouteRef2")
                    route2.setElement("EMSX_SEQUENCE",3745212)

                    route3 = routeRefIDPairs.appendElement()
                    route3.setElement("EMSX_ROUTE_REF_ID","MyRouteRef3")
                    route3.setElement("EMSX_SEQUENCE",3745213)

                    # Below we establish the strategy details. Strategy details
                    # are common across all orders in a GroupRoute operation.

                    strategy = request.getElement("EMSX_STRATEGY_PARAMS")
                    strategy.setElement("EMSX_STRATEGY_NAME", "VWAP")

                    indicator = strategy.getElement("EMSX_STRATEGY_FIELD_INDICATORS")
                    data = strategy.getElement("EMSX_STRATEGY_FIELDS")

                    # Strategy parameters must be appended in the correct order. See
↪the output
                    # of GetBrokerStrategyInfo request for the order. The indicator
↪value is 0 for
                    # a field that carries a value, and 1 where the field should be
↪ignored

                    data.appendElement().setElement("EMSX_FIELD_DATA", "09:30:00")
↪# StartTime
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "10:30:00")
↪# EndTime
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# Max%Volume
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# %AMSession
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# OPG
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# MOC
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# CompletePX
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# TriggerPX
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# DarkComplete
```

(continues on next page)

(continued from previous page)

```
                        indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                        data.appendElement().setElement("EMSX_FIELD_DATA", "")
# DarkCompPX
                        indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                        data.appendElement().setElement("EMSX_FIELD_DATA", "")
# RefIndex
                        indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                        data.appendElement().setElement("EMSX_FIELD_DATA", "")
# Discretion
                        indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                        print "Request: %s" % request.toString()

                        self.requestID = blpapi.CorrelationId()

                        session.sendRequest(request, correlationId=self.requestID )

                    elif msg.messageType() == SERVICE_OPEN_FAILURE:
                        print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 GroupRouteEx.py
Bloomberg - EMSX API Example - GroupRouteEx
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GroupRouteEx = {
    EMSX_SEQUENCE[] = {
        4747927, 4747928
    }
    EMSX_AMOUNT_PERCENT = 20
    EMSX_BROKER = "BB"
    EMSX_HAND_INSTRUCTION = "ANY"
    EMSX_ORDER_TYPE = MKT
    EMSX_TICKER = "GE US Equity"
    EMSX_TIF = DAY
    EMSX_ROUTE_REF_ID_PAIRS[] = {
        EMSX_ROUTE_REF_ID_PAIRS = {
            EMSX_ROUTE_REF_ID = "MyRouteRef1"
            EMSX_SEQUENCE = 4747927
        }
        EMSX_ROUTE_REF_ID_PAIRS = {
            EMSX_ROUTE_REF_ID = "MyRouteRef2"
            EMSX_SEQUENCE = 4747928
        }
```

(continues on next page)

(continued from previous page)

```
    }
}

Processing RESPONSE event
MESSAGE: GroupRouteEx = {
    EMSX_SUCCESS_ROUTES[] = {
        EMSX_SUCCESS_ROUTES = {
            EMSX_SEQUENCE = 4747927
            EMSX_ROUTE_ID = 1
        }
        EMSX_SUCCESS_ROUTES = {
            EMSX_SEQUENCE = 4747928
            EMSX_ROUTE_ID = 1
        }
    }
    EMSX_FAILED_ROUTES[] = {
    }
    MESSAGE = "2 of 2 Order(s) Routed"
    EMSX_ML_ID = "0:0"
}

CORRELATION ID: 3
MESSAGE TYPE: GroupRouteEx
SUCCESS: 4747927,1
SUCCESS: 4747928,1
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.19 Group Route Extended Request - Multi-Leg Options

The multi-leg options can be traded using `GroupRouteEx` request. The first step is to create the options and if need be equities leg using `CreateOrder` request. Once this is completed, create a request object for `GroupRouteEx` and submit it to the session with all the fields necessary for the multi-leg options routing.

The overall workflow for multi-leg options is similar to how you create and submit a basket or a list in EMSX.

The `CreateOrder` request will essentially stage the multi-leg options orders into EMSX. (e.g. B/O on AAPL US 11/20/15 C121 Equity and B/O on AAPL US 11/20/15 P119 Equity.)

The multi-leg request is an array and similar to submitting a basket order, it is important to make sure the `EMSX_SEQUENCE` matches in the `GroupRouteEx` with the orders created using `CreateOrder` request. For the subscription services, there will initially be eight elements to subscribe at the Route level subscription. They are `EMSX_ML_ID`, `EMSX_ML_LEG_QUANTITY`, `EMSX_ML_NUM_LEGS`, `EMSX_ML_PERCENT_FILLED`, `EMSX_ML_RATIO`, `EMSX_ML_REMAIN_BALANCE`, `EMSX_ML_STRATEGY`, and `EMSX_ML_TOTAL_QUANTITY`.

Please set the `EMSX_REQEST_TYPE` as `Multileg` to submit the multi-leg options using GroupRouteEx request.

Note: The Debit and Credit is indicated by the net price. Credit is indicated by using the negative sign in the net price where the Debit is indicated by the positive net price.

The net price can be specified using the EMSX_LIMIT_PRICE element for the multi-leg options orders.

Debit = positive for the net price

Credit = negative for the net price

3.15.20 Group Route Extended Request - Route As Spread

As of 15th of May, 2017 there also will be an ability to use GroupRouteEx to route two non-ticker as spread ticker in EMSX.

The underlying concept remains the same and the only difference is to use EMSX_REQUEST_TYPE as a spread instead of Multileg and for EMSX_TICKER use one of the two tickers that makes the spread ticker. The EMSX_SEQUENCE inside the array to submit the list remains the same for using GroupRouteEx to route as a spread.

Note: The EMSX_AMOUNT_PERCENT element for this request is used strictly for the amount in shares.

e.g. EMSX_AMOUNT_PERCENT, 100 means it'll send 100 shares from each ticker.

Full code sample:-

Route As Spread py

Hint: Please right click on the top code sample link to open in a new tab.

```
def routeSpread(self, session):

    request = self.service.createRequest("GroupRouteEx")

    request.append("EMSX_SEQUENCE", self.buySeqNo)
    request.append("EMSX_SEQUENCE", self.sellSeqNo)
    request.set("EMSX_AMOUNT_PERCENT", 100)
    request.set("EMSX_BROKER", "ETI");
    request.set("EMSX_HAND_INSTRUCTION", "ANY")
    request.set("EMSX_ORDER_TYPE", "MKT")
    request.set("EMSX_TIF", "DAY")
    request.set("EMSX_TICKER","CLN7 Comdty")
    request.set("EMSX_RELEASE_TIME",-1)
    requestType = request.getElement("EMSX_REQUEST_TYPE")
    requestType.setChoice("Spread")

    print "Request: %s" % request.toString()

    self.requestID = blpapi.CorrelationId()

    session.sendRequest(request, correlationId=self.requestID )
```

3.15.21 Manual Fill Request

The ManualFill request can be used on the sell-side EMSX<GO> settings to create fills and notifies EMSX<GO>.

Full code sample:-

Manual Fill cpp	Manual Fill cs	Manual Fill vba
Manual Fill java	Manual Fill py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("ManualFill");

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.set("EMSX_TRADER_UUID", 12109783)

            routeToFill = request.getElement("ROUTE_TO_FILL")

            routeToFill.setElement("EMSX_SEQUENCE", 1234567)
            routeToFill.setElement("EMSX_ROUTE_ID", 1)

            fills = request.getElement("FILLS")

            fills.setElement("EMSX_FILL_AMOUNT", 1000)
            fills.setElement("EMSX_FILL_PRICE", 123.4)
            fills.setElement("EMSX_LAST_MARKET", "XLON")

            fills.setElement("EMSX_INDIA_EXCHANGE","BGL")

            fillDateTime = fills.getElement("EMSX_FILL_DATE_TIME")

            fillDateTime.setChoice("Legacy");

            fillDateTime.setElement("EMSX_FILL_DATE",20172203)
            fillDateTime.setElement("EMSX_FILL_TIME",17054)
            fillDateTime.setElement("EMSX_FILL_TIME_FORMAT","SecondsFromMidnight")

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.15.22 Modify Order Extended Request

The `ModifyOrderEx` request modifies an existing or previously created order in EMSX<GO> or using EMSX API.

Important: Please note, when modifying an order or route, the limit price can be positive or negative. (e.g. Futures spreads). There are two special cases for setting the limit price to 0. In the EMSX_LIMIT_PRICE a value of 0 means to ignore the value. A value of EMSX_LIMIT_PRICE = -99999 means to reset the EMSX_LIMIT_PRICE to 0.

Full code sample:-

Modify Order Extended cpp	Modify Order Extended cs	Modify Order Extended vba
Modify Order Extended java	Modify Order Extended py	

Hint: Please right click on the top code sample link to open in a new tab.

```
     def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("ModifyOrderEx")

            # The fields below are mandatory
            request.set("EMSX_SEQUENCE", 3834157)
            request.set("EMSX_AMOUNT", 1300)
            request.set("EMSX_ORDER_TYPE", "MKT")
            request.set("EMSX_TIF", "DAY")
            request.set("EMSX_TICKER", "IBM US Equity")

            # The fields below are optional
            #request.set("EMSX_HAND_INSTRUCTION", "ANY")
            #request.set("EMSX_ACCOUNT","TestAccount")
            #request.set("EMSX_CFD_FLAG", "1")
            #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
            #request.set("EMSX_GET_WARNINGS", "0")
            #request.set("EMSX_GTD_DATE", "20170105")
            #request.set("EMSX_INVESTOR_ID", "InvID")
            #request.set("EMSX_LIMIT_PRICE", 123.45)
            #request.set("EMSX_NOTES", "Some notes")
            #request.set("EMSX_REQUEST_SEQ", 1001)
            #request.set("EMSX_STOP_PRICE", 123.5)

            # Note: When changing order type to a LMT order, you will need to␣
↪provide the EMSX_LIMIT_PRICE value.
            #       When changing order type away from LMT order, you will need to␣
↪reset the EMSX_LIMIT_PRICE value
            #       by setting the content to -99999

            # Note: To clear down the stop price, set the content to -1

            # If modifying on behalf of another trader, set the order owner's UUID
            #request.set("EMSX_TRADER_UUID", 1234567)
```

(continues on next page)

(continued from previous page)

```
41
42              print "Request: %s" % request.toString()
43
44              self.requestID = blpapi.CorrelationId()
45
46              session.sendRequest(request, correlationId=self.requestID )
47
48          elif msg.messageType() == SERVICE_OPEN_FAILURE:
49              print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 ModifyOrder.py
Bloomberg - EMSX API Example - ModifyOrderEx
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: ModifyOrderEx = {
    EMSX_SEQUENCE = 4747927
    EMSX_AMOUNT = 6000
    EMSX_ORDER_TYPE = MKT
    EMSX_TIF = DAY
    EMSX_TICKER = "MSFT US Equity"
    EMSX_INVESTOR_ID = "InvID"
}

Processing RESPONSE event
MESSAGE: ModifyOrderEx = {
    EMSX_SEQUENCE = 4747927
    MESSAGE = "Order Modified"
}

CORRELATION ID: 3
MESSAGE TYPE: ModifyOrderEx
EMSX_SEQUENCE: 4747927  MESSAGE: Order Modified
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.23 Modify Route Extended Request

The `ModifyRouteEx` request modifies an existing or previously created child routes in EMSX<GO> or using EMSX API.

Important: Please note, when modifying an order or route, the limit price can be positive or negative. (e.g. Futures spreads). There are two special cases for setting the limit price to 0. In the EMSX_LIMIT_PRICE a value of 0 means to ignore the value. A value of EMSX_LIMIT_PRICE = -99999 means to reset the EMSX_LIMIT_PRICE to 0.

Full code sample:-

Modify Route Extended cpp	Modify Route Extended cs	Modify Route Extended vba
Modify Route Extended java	Modify Route Extended py	

Hint: Please right click on the top code sample link to open in a new tab.

```
     def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:

            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("ModifyRouteEx")

            # The fields below are mandatory
            request.set("EMSX_SEQUENCE", 3834157)
            request.set("EMSX_ROUTE_ID", 1)
            request.set("EMSX_AMOUNT", 1000)
            request.set("EMSX_ORDER_TYPE", "MKT")
            request.set("EMSX_TIF", "DAY")

            # The fields below are optional
            #request.set("EMSX_ACCOUNT","TestAccount")
            #request.set("EMSX_CLEARING_ACCOUNT", "ClearingAcnt")
            #request.set("EMSX_CLEARING_FIRM", "ClearingFirm")
            #request.set("EMSX_COMM_TYPE", "Absolute")
            #request.set("EMSX_EXCHANGE_DESTINATION", "DEST")
            #request.set("EMSX_GET_WARNINGS", "0")
            #request.set("EMSX_GTD_DATE", "20170105")
            #request.set("EMSX_LIMIT_PRICE", 123.45)
            #request.set("EMSX_LOC_BROKER", "ABCD")
            #request.set("EMSX_LOC_ID", "1234567")
            #request.set("EMSX_LOC_REQ", "Y")
            #request.set("EMSX_NOTES", "Some notes")
            #request.set("EMSX_ODD_LOT", "" )
            #request.set("EMSX_P_A", "P")
            #request.set("EMSX_REQUEST_SEQ", 1001)
            #request.set("EMSX_STOP_PRICE", 123.5)
            #request.set("EMSX_TRADER_NOTES", "Trader notes")
            #request.set("EMSX_USER_COMM_RATE", 0.02)
            #request.set("EMSX_USER_FEES", "1.5")

            # Note: When changing order type to a LMT order, you will need to␣
↪provide the EMSX_LIMIT_PRICE value.
```

(continues on next page)

(continued from previous page)

```
            #       When changing order type away from LMT order, you will need to reset the EMSX_LIMIT_PRICE value
            #       by setting the content to -99999

            # Note: To clear down the stop price, set the content to -1

            # Set the strategy parameters, if required

            '''
            strategy = request.getElement("EMSX_STRATEGY_PARAMS")
            strategy.setElement("EMSX_STRATEGY_NAME", "VWAP")

            indicator = strategy.getElement("EMSX_STRATEGY_FIELD_INDICATORS")
            data = strategy.getElement("EMSX_STRATEGY_FIELDS")

            # Strategy parameters must be appended in the correct order. See the output
            # of GetBrokerStrategyInfo request for the order. The indicator value is 0 for
            # a field that carries a value, and 1 where the field should be ignored

            data.appendElement().setElement("EMSX_FIELD_DATA", "09:30:00") # StartTime
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

            data.appendElement().setElement("EMSX_FIELD_DATA", "10:30:00") # EndTime
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # Max %Volume
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # %AMSession
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # OPG
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # MOC
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # CompletePX
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # TriggerPX
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # DarkComplete
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

            data.appendElement().setElement("EMSX_FIELD_DATA", "")         # DarkCompPX
            indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
```

(continues on next page)

(continued from previous page)

```
90
91              data.appendElement().setElement("EMSX_FIELD_DATA", "")          # RefIndex
92              indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
93
94              data.appendElement().setElement("EMSX_FIELD_DATA", "")          #␣
    ↪Discretion
95              indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
96              '''
97
98              # If modifying on behalf of another trader, set the order owner's UUID
99              #request.set("EMSX_TRADER_UUID", 1234567)
100
101             # If modifying a multi-leg route, indicate the Multileg ID
102             #request.getElement("EMSX_REQUEST_TYPE").setChoice("Multileg").
    ↪setElement("EMSX_ML_ID", "123456")
103
104             print "Request: %s" % request.toString()
105
106             self.requestID = blpapi.CorrelationId()
107
108             session.sendRequest(request, correlationId=self.requestID )
109
110         elif msg.messageType() == SERVICE_OPEN_FAILURE:
111             print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 ModifyRouteEx.py
Bloomberg - EMSX API Example - ModifyRouteEx
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: ModifyRouteEx = {
    EMSX_SEQUENCE = 4747928
    EMSX_ROUTE_ID = 1
    EMSX_AMOUNT = 500
    EMSX_ORDER_TYPE = MKT
    EMSX_TIF = DAY
}

Processing RESPONSE event
MESSAGE: ModifyRouteEx = {
    EMSX_SEQUENCE = 0
    EMSX_ROUTE_ID = 0
    MESSAGE = "Route modified"
}

CORRELATION ID: 3
MESSAGE TYPE: ModifyRouteEx
```

(continues on next page)

(continued from previous page)

```
MESSAGE: Route modified
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.24 Route Extended Request

The `RouteEx` request submits an existing order into various execution veneues. This request is used primarily to submit a child route based on previously created parent order.

Full code sample:-

Route Extended cpp	Route Extended cs	Route Extended vba
Route Extended java	Route Extended py	

Hint: Please right click on the top code sample link to open in a new tab.

```
        def processServiceStatusEvent(self,event,session):
            print "Processing SERVICE_STATUS event"

            for msg in event:

                if msg.messageType() == SERVICE_OPENED:
                    print "Service opened..."

                    service = session.getService(d_service)

                    request = service.createRequest("RouteEx")

                    # The fields below are mandatory
                    request.set("EMSX_SEQUENCE", 3745217) # Order number
                    request.set("EMSX_AMOUNT", 500)
                    request.set("EMSX_BROKER", "BB")
                    request.set("EMSX_HAND_INSTRUCTION", "ANY")
                    request.set("EMSX_ORDER_TYPE", "MKT")
                    request.set("EMSX_TICKER", "IBM US Equity")
                    request.set("EMSX_TIF", "DAY")

                    # The fields below are optional
                    #request.set("EMSX_ACCOUNT","TestAccount")
                    ##request.set("EMSX_CFD_FLAG", "1")
                    #request.set("EMSX_CLEARING_ACCOUNT", "ClrAccName")
                    #request.set("EMSX_CLEARING_FIRM", "FirmName")
                    #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
                    #request.set("EMSX_GET_WARNINGS", "0")
                    #request.set("EMSX_GTD_DATE", "20170105")
                    #request.set("EMSX_LIMIT_PRICE", 123.45)
```

(continues on next page)

(continued from previous page)

```
                        #request.set("EMSX_LOCATE_BROKER", "BMTB")
                        #request.set("EMSX_LOCATE_ID", "SomeID")
                        #request.set("EMSX_LOCATE_REQ", "Y")
                        #request.set("EMSX_NOTES", "Some notes")
                        #request.set("EMSX_ODD_LOT", "0")
                        #request.set("EMSX_P_A", "P")
                        #request.set("EMSX_RELEASE_TIME", 1259)
                        #request.set("EMSX_REQUEST_SEQ", 1001)
                        #request.set("EMSX_ROUTE_REF_ID", "UniqueRef")
                        #request.set("EMSX_STOP_PRICE", 123.5)
                        #request.set("EMSX_TRADER_UUID", 1234567)

                        print "Request: %s" % request.toString()

                        self.requestID = blpapi.CorrelationId()

                        session.sendRequest(request, correlationId=self.requestID )

                    elif msg.messageType() == SERVICE_OPEN_FAILURE:
                        print >> sys.stderr, "Error: Service failed to open"
```

Output:-

```
C:\Users\_scripts>py -3 RouteEx.py
Bloomberg - EMSX API Example - RouteWithStrat
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
    server = "localhost:8194"
    encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: RouteEx = {
    EMSX_SEQUENCE = 4747927
    EMSX_AMOUNT = 200
    EMSX_BROKER = "BB"
    EMSX_HAND_INSTRUCTION = "ANY"
    EMSX_ORDER_TYPE = MKT
    EMSX_TICKER = "MSFT US Equity"
    EMSX_TIF = DAY
    EMSX_NOTES = "Some notes"
    EMSX_P_A = "P"
}

Processing RESPONSE event
MESSAGE: Route = {
    EMSX_SEQUENCE = 4747927
    EMSX_ROUTE_ID = 2
    MESSAGE = "Order Routed"
}

CORRELATION ID: 3
MESSAGE TYPE: Route
```

(continues on next page)

(continued from previous page)

```
EMSX_SEQUENCE: 4747927  EMSX_ROUTE_ID: 2        MESSAGE: Order Routed
Processing SESSION_STATUS event
SessionConnectionDown = {
    server = "localhost:8194"
}

Processing SESSION_STATUS event
SessionTerminated = {
}
```

3.15.25 Route Manually Extended Request

The `RouteManuallyEx` requestis generally used for phone orders where the placement is external to EMSX API. This request creates an order and notifies EMSX<GO> that this order is routed to the execution venue.

Full code sample:-

Route Manually cpp	Route Manually cs	Route Manually vba
Route Manually java	Route Manually py	

Hint: Please right click on the top code sample link to open in a new tab.

```
        def processServiceStatusEvent(self,event,session):
            print "Processing SERVICE_STATUS event"

            for msg in event:

                if msg.messageType() == SERVICE_OPENED:
                    print "Service opened..."

                    service = session.getService(d_service)

                    request = service.createRequest("RouteManuallyEx")

                    # The fields below are mandatory
                    request.set("EMSX_SEQUENCE", 3745218)  # Order number
                    request.set("EMSX_AMOUNT", 500)
                    request.set("EMSX_BROKER", "BB")
                    request.set("EMSX_HAND_INSTRUCTION", "ANY")
                    request.set("EMSX_ORDER_TYPE", "MKT")
                    request.set("EMSX_TICKER", "IBM US Equity")
                    request.set("EMSX_TIF", "DAY")

                    # The fields below are optional
                    #request.set("EMSX_ACCOUNT","TestAccount")
                    #request.set("EMSX_BOOKNAME","BookName")
                    #request.set("EMSX_CFD_FLAG", "1")
                    #request.set("EMSX_CLEARING_ACCOUNT", "ClrAccName")
                    #request.set("EMSX_CLEARING_FIRM", "FirmName")
                    #request.set("EMSX_EXEC_INSTRUCTIONS", "AnyInst")
                    #request.set("EMSX_GET_WARNINGS", "0")
                    #request.set("EMSX_GTD_DATE", "20170105")
```

(continues on next page)

(continued from previous page)

```
                    #request.set("EMSX_LIMIT_PRICE", 123.45)
                    #request.set("EMSX_LOCATE_BROKER", "BMTB")
                    #request.set("EMSX_LOCATE_ID", "SomeID")
                    #request.set("EMSX_LOCATE_REQ", "Y")
                    #request.set("EMSX_NOTES", "Some notes")
                    #request.set("EMSX_ODD_LOT", "0")
                    #request.set("EMSX_P_A", "P")
                    #request.set("EMSX_RELEASE_TIME", 1259)
                    #request.set("EMSX_REQUEST_SEQ", 1001)
                    #request.set("EMSX_ROUTE_REF_ID", "UniqueRef")
                    #request.set("EMSX_STOP_PRICE", 123.5)
                    #request.set("EMSX_TRADER_UUID", 1234567)

                    # Below we establish the strategy details
                    '''
                    strategy = request.getElement("EMSX_STRATEGY_PARAMS")
                    strategy.setElement("EMSX_STRATEGY_NAME", "VWAP")

                    indicator = strategy.getElement("EMSX_STRATEGY_FIELD_INDICATORS")
                    data = strategy.getElement("EMSX_STRATEGY_FIELDS")

                    # Strategy parameters must be appended in the correct order. See␣
↪the output
                    # of GetBrokerStrategyInfo request for the order. The indicator␣
↪value is 0 for
                    # a field that carries a value, and 1 where the field should be␣
↪ignored

                    data.appendElement().setElement("EMSX_FIELD_DATA", "09:30:00")
↪# StartTime
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "10:30:00")
↪# EndTime
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 0)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# Max%Volume
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# %AMSession
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# OPG
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# MOC
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# CompletePX
                    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)

                    data.appendElement().setElement("EMSX_FIELD_DATA", "")
↪# TriggerPX
```

(continues on next page)

(continued from previous page)

```
78                      indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
79
80                      data.appendElement().setElement("EMSX_FIELD_DATA", "")
    ↪# DarkComplete
81                      indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
82
83                      data.appendElement().setElement("EMSX_FIELD_DATA", "")
    ↪# DarkCompPX
84                      indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
85
86                      data.appendElement().setElement("EMSX_FIELD_DATA", "")
    ↪# RefIndex
87                      indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
88
89                      data.appendElement().setElement("EMSX_FIELD_DATA", "")
    ↪# Discretion
90                      indicator.appendElement().setElement("EMSX_FIELD_INDICATOR", 1)
91                      '''
92
93                      print "Request: %s" % request.toString()
94
95                      self.requestID = blpapi.CorrelationId()
96
97                      session.sendRequest(request, correlationId=self.requestID )
98
99                  elif msg.messageType() == SERVICE_OPEN_FAILURE:
100                     print >> sys.stderr, "Error: Service failed to open"
```

3.16 Sell-Side Request/Response Service

The sell-side Request/Response service is specifically used for EMSX to EMSX (E2E) setting where the sell-side EMSX is used to capture order flow from other buy-side EMSX users.

The EMSX API allows developers to use the Request/Response services for order and route creation, modification, queries related to orders and routes (placements) as well as EMSX Team details. Depending on the type of action required, the application programmer must create a specific request, populate it with required parameters and send that request to the EMSX API service, which provides the response. Communication with the request/response service requires the following steps:

1. Create a session (if session does not yet exist).
2. Connect session to `//blp/emapisvc_beta` or `//blp/emapisvc` service and start it.
3. Fetch a service object from the session representing emapisvc.
4. Use the service object from above to create a Request object of the desired type
5. Send request object via sendRequest method of session object, pass object of type EventQueue to the sendRequest.
6. Loop through the EventQueue object until event of type `Event::RESPONSE` is read.

These are initialized in the constructor as below and are then available for the life of the application for submission of various requests.

3.16.1 Manual Fill Request

The `ManualFill` request can be used on the sell-side EMSX<GO> settings to create fills and notifies EMSX<GO>.

Full code sample:-

Manual Fill cpp	Manual Fill cs	Manual Fill vba
Manual Fill java	Manual Fill py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("ManualFill");

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.set("EMSX_TRADER_UUID", 12109783)

            routeToFill = request.getElement("ROUTE_TO_FILL")

            routeToFill.setElement("EMSX_SEQUENCE", 1234567)
            routeToFill.setElement("EMSX_ROUTE_ID", 1)

            fills = request.getElement("FILLS")

            fills.setElement("EMSX_FILL_AMOUNT", 1000)
            fills.setElement("EMSX_FILL_PRICE", 123.4)
            fills.setElement("EMSX_LAST_MARKET", "XLON")

            fills.setElement("EMSX_INDIA_EXCHANGE","BGL")

            fillDateTime = fills.getElement("EMSX_FILL_DATE_TIME")

            fillDateTime.setChoice("Legacy");

            fillDateTime.setElement("EMSX_FILL_DATE",20172203)
            fillDateTime.setElement("EMSX_FILL_TIME",17054)
            fillDateTime.setElement("EMSX_FILL_TIME_FORMAT","SecondsFromMidnight")

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
```

(continues on next page)

(continued from previous page)

```
45              print >> sys.stderr, "Error: Service failed to open"
```

3.16.2 Sell Side Ack Request

The SellSideAck request is used on the sell-side EMSX<GO> settings to create Ack message on incoming orders from buy-side EMSX<GO>.

Full code sample:-

Sell Side Ack cpp	Sell Side Ack cs	Sell Side Ack vba
Sell Side Ack java	Sell Side Ack py	

Hint: Please right click on the top code sample link to open in a new tab.

```
1  def processServiceStatusEvent(self,event,session):
2      print "Processing SERVICE_STATUS event"
3
4      for msg in event:
5
6          if msg.messageType() == SERVICE_OPENED:
7              print "Service opened..."
8
9              service = session.getService(d_service)
10
11             request = service.createRequest("SellSideAck");
12
13             #request.set("EMSX_REQUEST_SEQ", 1)
14
15             request.append("EMSX_SEQUENCE", 1234567)
16
17             # The following Element is currently not being used in this request.
18             #request.set("EMSX_TRADER_UUID", 7654321)
19
20             print "Request: %s" % request.toString()
21
22             self.requestID = blpapi.CorrelationId()
23
24             session.sendRequest(request, correlationId=self.requestID )
25
26         elif msg.messageType() == SERVICE_OPEN_FAILURE:
27             print >> sys.stderr, "Error: Service failed to open"
```

3.16.3 Sell Side Reject Request

The SellSideReject request is used on the sell-side EMSX<GO> settings to create Reject message on incoming orders from buy-side EMSX<GO>.

Full code sample:-

Sell Side Reject cpp	Sell Side Reject cs	Sell Side Reject vba
Sell Side Reject java	Sell Side Reject py	

Hint: Please right click on the top code sample link to open in a new tab.

```
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."

            service = session.getService(d_service)

            request = service.createRequest("SellSideReject");

            #request.set("EMSX_REQUEST_SEQ", 1)

            request.append("EMSX_SEQUENCE", 1234567)

            # The following Element is currently not being used in this request.
            #request.set("EMSX_TRADER_UUID", 7654321)

            print "Request: %s" % request.toString()

            self.requestID = blpapi.CorrelationId()

            session.sendRequest(request, correlationId=self.requestID )

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"
```

3.17 EMSX Subscription

EMSX subscription service provides a way of accessing and monitoring real-time updates on orders and routes in the user's blotter outside of EMSX<GO> function in your Bloomberg terminal.

EMSX subscription sample illustrates how to use both Order and Route subscription service for EMSX API.

Once the subscription is established all the orders and routes in the user's blotter are returned via one or more Bloomberg API events of type SUBSCRIPTION. Each event is further composed of one or more messages where each message contains all the subscribed fields for a single order or route.

Additionally, any changes to these orders and/or routes will generate events that are passed along as they occur. These subscriptions can be asynchronous or synchronous but it is best to always approach this with asynchronous event-driven architecture in mind.

Warning: When implementing subscription service, it's important to write the code using two separate .subscribe() events for the order and route subscriptions.

Important: It's important to unsubscribe before starting the Subscription service.

3.17.1 Description of Subscription Messages

Element Name	Description
MSG_TYPE	MSG_TYPE=E, this indicates the message is an EMSX API message.
MSG_SUB_TYPE	O = Order & R = Route
EVENT_STATUS	Event status messages (e.g INIT_PAINT, NEW_ORDER_ROUTE and etc.)
API_SEQ_NUM	Unique API sequence number to help detect gaps in the events.
EMSX_SEQUENCE	Unique order number in EMSX<GO>.
EMSX_ROUTE_ID	Route number, always 0 for order subscription events.
EMSX_FILL_ID	Fill number on routess.

3.17.2 Description of Event Status Messages

EVENT_STATUS	Message Type / Description
EVENT_STATUS = 1	Heartbeat Message HB_MESSAGE
EVENT_STATUS = 4	Initial Paint Message on all subscription fields INIT_PAINT
EVENT_STATUS = 6	New Order or Route Message on all subscription fields NEW_ORDER_ROUTE
EVENT_STATUS = 7	This field dynamically updates for existing Order and route UPD_ORDER_ROUTE
EVENT_STATUS = 8	Order and route deletion message, DELETION_MESSAGE
EVENT_STATUS = 11	The end of the initial paint message, INIT_PAINT_END

3.17.3 Description of Order Status Messages

Order Status	Description
ASSIGN	The route has been cancelled or rejected without fills.
	Applicable Child Route Status: CANCEL or REJECTED.
CANCEL	The order has been cancelled, no shares filled.
	Applicable Child Route Status: CANCEL or REJECTED.
COMPLETED	All Shares have been filled and allocated in OAX for Bloomberg AIM users.
	Applicable Child Route Status: CANCEL, FILLED, or PARTFILLED.
CXL-PEND	The Sell-Side EMSX to EMSX (E2E), order pending cancel acknowledgement.
EXPIRED	The order is expired.
	Applicable Child Route Status: CANCEL, FILLED, or PARTFILLED.
FILLED	All shares have been filled, no idle quantity.
	Applicable Child Route Status: FILLED.
MOD-PEND	Only valid for the Sell-Side EMSX to EMSX (E2E) settings. The order modification
	pending acknowledgement. Fields that can populate: Size, Price, Stop, GTDDate,
	TIF, Type and instruments. e.g. EMSX_MOD_PEND_STATUS= "Pending Info|Size: 500.0
	-> 200.0|Price 2.0000 -> 4.0000|Instr: -> test instr"
NEW	The order has been added/staged; no routes have been created.
ORD-PEND	The Sell-Side EMSX to EMSX (E2E), new order pending acknowledgement.
PARTFILLED	The order has idle or unfilled shares.
	Applicable Child Route Status: CANCEL, FILLED, or PARTFILLED.
SENT	The route has been sent to the broker but has not been acknowledged.
	Applicable Child Route Status: SENT.
WORKING	The route has been sent and acknowledged by the broker or the route has been
	partially filled or route has a cancel request pending or rejected.
	Applicable Child Route Status: CXLREJ, CXLREQ, CXLRPRQ, CXLRPRJ,
	HOLD, PARTFILLED, or WORKING

3.17.4 Description of the Child Route Status Messages

Route Status	Description
A-SENT	The route has been sent for allocation for Bloomberg STP users.
ALLOCATED	The route has been allocated for Bloomberg STP users.
BUST	The route fill has been busted by the execution broker.
CANCEL	The route has been canceled.
CORRECTED	The route fill has been corrected by the execution broker.
CXLREJ	The cancel request is rejected by the execution broker.
CXLREP	The cancel replace request is accepted by the execution broker.
CXLREQ	The cancel request is sent and is pending with the execution broker.
CXLRPRJ	The cancel replace request is rejected by the execution broker.
CXLRPRQ	The cancel replace request is sent and is pending with the execution broker.
DONE	The route has been marked done for the day by the execution broker.
FILLED	The route has been completely filled.
HOLD	The shared are committed to a dark pool.
OA-SENT	The route has been sent for allocation in OAX for Bloomberg AIM users
OMS PEND	The route has been sent to buy-side OMS for compliance check, pending acknowledgement.
PARTFILLED	The route has been partilly filled.
QUEUED	The route is created but not released until the defined time in release time.
REJECTED	The route has been rejected by the execution broker.
REPPEN	The route replace request is pending with the execution broker.
ROUTE-ERR	The route has an error, please check with EMSX trade desk and/or executing broker.
SENT	The route has been sent to the broker but have not been acknowledged by the broker.
WORKING	The route has been sent and acknowledged by the executing broker.

3.17.5 Description of the Child Route Status Changes

Field	Previous Value	New Value	Definition
EMSX_STATUS	null	SENT	New route (placement) created.
EMSX_STATUS	SENT	SENT	Field update on sent.
EMSX_STATUS	SENT	WORKING	ACK received from the broker.
EMSX_STATUS	WORKING	PARTFILL	First fill or multiple fills.
EMSX_WORKING	n	<n and >0	(<100%)
EMSX_STATUS	PARTFILL	PARTFILL	Middle fill or multiple
EMSX_WORKING	n	<n and >0	fills. (<100%)
EMSX_STATUS	PARTFILL	FILLED	Final fill or multiple fills.
EMSX_WORKING	>0	0	(100%)
EMSX_STATUS	WORKING	FILLED	Full single fill.
EMSX_WORKING	>0	0	
EMSX_STATUS	null	FILLED	Historic 100% fill on INIT_PAINT.
EMSX_STATUS	null	WORKING	Working route (placement) on INIT_PAINT.
EMSX_STATUS	null	PARTFILL	Part filled route (placement) on INIT_PAINT.
EMSX_STATUS	null	CXLREQ	Cancel requested on route in INIT_PAINT.
EMSX_STATUS	WORKING	CXLREQ	Cancel route request sent.
EMSX_STATUS	CXLREQ	WORKING	Broker rejected cancel request.
EMSX_STATUS	CXLREQ	CXLPEN	Broker sent ACK for cancel request.
EMSX_STATUS	CXLPEN	WORKING	Broker rejected cancel request.
EMSX_STATUS	CXLREQ	CANCEL	Broker cancelled route from request.

Continued on next page

Table 1 – continued from previous page

Field	Previous Value	New Value	Definition
EMSX_STATUS	CXLPEN	CANCEL	Broker cancelled route from request.
EMSX_STATUS	PARTFILL	CXLREQ	Cancel requested on part filled route.
EMSX_STATUS	CXLREQ	PARTFILL	Broker rejected cancel request.
EMSX_STATUS	CXLPEN	PARTFILL	Broker rejected cancel request.
EMSX_STATUS	WORKING	CXLRPRQ	Modify (cancel/replace) request sent to broker.
EMSX_STATUS	CXLRPRQ	REPPEN	Broker sent ACK for modify request.
EMSX_STATUS	REPPEN	WORKING	Broker rejected modify request on working route.
EMSX_BROKER_STATUS	n/a	CXLRPRJ	
EMSX_STATUS	REPPEN	WORKING	Broker accepted and applied the modify request on working route. (placement)
EMSX_BROKER_STATUS	n/a	MODIFIED	
EMSX_STATUS	PARTFILL	CXLRPRQ	Modify (cancel/replace) request sent to broker.
EMSX_STATUS	REPPEN	PARTFILL	Broker rejected modify request on part filled route. (placement)
EMSX_BROKER_STATUS	n/a	CXLRPRJ	
EMSX_STATUS	REPPEN	PARTFILL	Broker accepted and applied the modify request on part filled route. (placement)
EMSX_BROKER_STATUS	n/a	MODIFIED	
EMSX_STATUS	SENT	REJECTED	Broker rejected the order from sent status.
EMSX_STATUS	null	REJECTED	INIT_PAINT shows route (placement) rejected.
EMSX_STATUS	null	CANCEL	INIT_PAINT shows route (placement) cancelled.
EMSX_STATUS	CXLRPRQ	WORKING	Modify rejected from request.
EMSX_STATUS	PARTFILL	CANCEL	Part filled route cancelled by broker.
EMSX_STATUS	WORKING	CANCEL	Working route cancelled by broker.
EMSX_STATUS	WORKING	REJECTED	Route rejected from working.

3.17.6 Description of Fills using Route Subscription

The real-time fills in EMSX API are delivered through the route subscription service. However, to capture the full state of the order, we always recommend the client listens to both the order and route subscription service.

The following elements provide the route updates that can be calculated to obtain the real-time incoming fills for a live route.

Field	Definition
EMSX_LAST_FILL_DATE	INT32 ROUTE The date of the last fill based on the user's time zone. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_LAST_MARKET	STRING ROUTE The last market of execution for a trade as returned by the broker.This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_LAST_PRICE	FLOAT64 ROUTE The last execution price for a trade. This field is applicable to trades on a route level, and does not populate ona per security basis.
EMSX_LAST_SHARES	INT32 ROUTE The last executed quantity for a trade. This field is applicable to trades on a route level, and does not populate on a per security basis.
EMSX_LAST_FILL_TIME	INT32 ROUTE The time of the last fill based on seconds from midnight in the user's time zone. This field is applicable to trades on a route level, and does not populate on a per security basis.

The EMSX_FILL_ID is the transaction sequence number to keep track of the individual fills. One thing to keep in mind is that this is a reflection of the fills and thus you will typically see the EMSX_FILL_ID to show 0, 2, 3, 4,.. 8,9,.. 14, and etc. In most cases, the EMSX_FILL_ID = 1 is not reflected as this is an ACK message from the broker. The EMSX_FILL_ID is a unique ID per fill in sequential order but does not necessarily tie to the actual Fill numbers and will skip fill events that are not directly tied to a fill.

Field	Definition
EMSX_FILL_ID	INT32 STATIC O,R The fill number associated with a route. This field is applicable to trades on an order and/or route level, and does not populate on a per security basis.

The EMSX_ROUTE_LAST_UPDATE_TIME is timestamp based on the number of seconds from midnight that reflects the last update of a route. This can be fill or any other route-based update events.

Field	Definition
EMSX_ROUTE_LAST_UPDATE_TIME	INT32 ROUTE The time stamp of the last execution or cancellation on a route. This field is applicable to trades on a route level and does not populate on a per security basis.

3.17.7 Description of Order Expiration Logic

The parent orders in EMSX follow an expiration logic that first puts orders into view only mode before it gets removed from EMSX blotter.

Note: TIF = Time in force

h = hours

GT covers both GTC and GTD.

Asset	TIF	Event	Description
Equity	Day	EXPIRED	Exchange close + 8h
Equity	Day	DELETED	Exchange close + 8h + 16h
Equity	GT	EXPIRED	On GTD date it's same as day order if there are no open routes
Equity	GT	EXPIRED	On GTD date if open routes, then redated to current GTD date + 24h
Future	Day	EXPIRED	Earlier of Exchange close + 4h or start of the next session
Future	Day	DELETED	Earlier of Exchange close + 4h or start of the next session + 20h
Future	GT	EXPIRED	On GTD date it's same as day order if there are no open routes
Future	GT	EXPIRED	On GTD date if open routes, then redated to current GTD date + 24h
Option	Day	EXPIRED	Exchange close + 4h
Option	Day	DELETED	Exchange close + 4h + 20h
Option	GT	EXPIRED	On GTD date it's same as day order if there are no open routes.
Option	GT	EXPIRED	On GTD date if open routes, then redated to current GTD date + 24h

3.17.8 Description of Route Expiration Logic

All equities routes in EMSX will expire 8 hours after the exchange midnight. All futures and options routes in EMSX will expire 24 hours after exchange close time.

Full code sample:-

EMSX Subscriptions cpp	EMSX Subscriptions cs	EMSX Subscription vba
EMSX Subscriptions java	EMSX Subscriptions py	EMSX Subscriptions py2

Hint: Please right click on the top code sample link to open in a new tab.

Specify service name and host/port :-

```
# EMSXSubscriptions.py

import blpapi
import sys


ORDER_ROUTE_FIELDS              = blpapi.Name("OrderRouteFields")

SLOW_CONSUMER_WARNING           = blpapi.Name("SlowConsumerWarning")
SLOW_CONSUMER_WARNING_CLEARED   = blpapi.Name("SlowConsumerWarningCleared")

SESSION_STARTED                 = blpapi.Name("SessionStarted")
SESSION_TERMINATED              = blpapi.Name("SessionTerminated")
SESSION_STARTUP_FAILURE         = blpapi.Name("SessionStartupFailure")
SESSION_CONNECTION_UP           = blpapi.Name("SessionConnectionUp")
SESSION_CONNECTION_DOWN         = blpapi.Name("SessionConnectionDown")

SERVICE_OPENED                  = blpapi.Name("ServiceOpened")
SERVICE_OPEN_FAILURE            = blpapi.Name("ServiceOpenFailure")

SUBSCRIPTION_FAILURE            = blpapi.Name("SubscriptionFailure")
SUBSCRIPTION_STARTED            = blpapi.Name("SubscriptionStarted")
SUBSCRIPTION_TERMINATED         = blpapi.Name("SubscriptionTerminated")

EXCEPTIONS = blpapi.Name("exceptions")
FIELD_ID = blpapi.Name("fieldId")
REASON = blpapi.Name("reason")
CATEGORY = blpapi.Name("category")
DESCRIPTION = blpapi.Name("description")

d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
orderSubscriptionID=blpapi.CorrelationId(98)
routeSubscriptionID=blpapi.CorrelationId(99)
```

Process admin events:-

```
def processAdminEvent(self,event):
    print "Processing ADMIN event"
```

(continues on next page)

(continued from previous page)

```
    for msg in event:

        if msg.messageType() == SLOW_CONSUMER_WARNING:
            print "Warning: Entered Slow Consumer status"
        elif msg.messageType() ==  SLOW_CONSUMER_WARNING_CLEARED:
            print "Slow consumer status cleared"

def processSessionStatusEvent(self,event,session):
    print "Processing SESSION_STATUS event"

    for msg in event:

        if msg.messageType() == SESSION_STARTED:
            print "Session started..."
            session.openServiceAsync(d_service)

        elif msg.messageType() == SESSION_STARTUP_FAILURE:
            print >> sys.stderr, "Error: Session startup failed"

        elif msg.messageType() == SESSION_TERMINATED:
            print >> sys.stderr, "Error: Session has been terminated"

        elif msg.messageType() == SESSION_CONNECTION_UP:
            print "Session connection is up"

        elif msg.messageType() == SESSION_CONNECTION_DOWN:
            print >> sys.stderr, "Error: Session connection is down"

def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

        if msg.messageType() == SERVICE_OPENED:
            print "Service opened..."
            self.createOrderSubscription(session)

        elif msg.messageType() == SERVICE_OPEN_FAILURE:
            print >> sys.stderr, "Error: Service failed to open"

def processSubscriptionStatusEvent(self, event, session):
    print "Processing SUBSCRIPTION_STATUS event"
```

Start Subscription:-

```
for msg in event:

    if msg.messageType() == SUBSCRIPTION_STARTED:

        print "OrderSubID: %s\tRouteSubID: %s" % (orderSubscriptionID.value(),␣
↪routeSubscriptionID.value())

        if msg.correlationIds()[0].value() == orderSubscriptionID.value():
            print "Order subscription started successfully"
            self.createRouteSubscription(session)

        elif msg.correlationIds()[0].value() == routeSubscriptionID.value():
```

(continues on next page)

(continued from previous page)

```
            print "Route subscription started successfully"

    elif msg.messageType() == SUBSCRIPTION_FAILURE:
        print >> sys.stderr, "Error: Subscription failed"
        print >> sys.stderr, "MESSAGE: %s" % (msg)

        reason = msg.getElement("reason");
        errorcode = reason.getElementAsInteger("errorCode")
        description = reason.getElementAsString("description")

        print >> sys.stdout, "Error: (%d) %s" % (errorcode, description)

    elif msg.messageType() == SUBSCRIPTION_TERMINATED:
        print >> sys.stderr, "Error: Subscription terminated"
        print >> sys.stderr, "MESSAGE: %s" % (msg)
```

Pick and choose the elements and create order subscription:-

```
def createOrderSubscription(self, session):

    print "Create Order subscription"

    orderTopic = d_service + "/order?fields="
    orderTopic = orderTopic + "API_SEQ_NUM,"
    orderTopic = orderTopic + "EMSX_ACCOUNT,"
    orderTopic = orderTopic + "EMSX_AMOUNT,"
    orderTopic = orderTopic + "EMSX_ASSET_CLASS,"
    orderTopic = orderTopic + "EMSX_ASSIGNED_TRADER,"
    orderTopic = orderTopic + "EMSX_AVG_PRICE,"
    orderTopic = orderTopic + "EMSX_BASKET_NAME,"
    orderTopic = orderTopic + "EMSX_BASKET_NUM,"
    orderTopic = orderTopic + "EMSX_BROKER,"
    orderTopic = orderTopic + "EMSX_BROKER_COMM,"
    orderTopic = orderTopic + "EMSX_BSE_AVG_PRICE,"
    orderTopic = orderTopic + "EMSX_BSE_FILLED,"
    orderTopic = orderTopic + "EMSX_CFD_FLAG,"
    orderTopic = orderTopic + "EMSX_COMM_DIFF_FLAG,"
    orderTopic = orderTopic + "EMSX_COMM_RATE,"
    orderTopic = orderTopic + "EMSX_CURRENCY_PAIR,"
    orderTopic = orderTopic + "EMSX_DATE,"
    orderTopic = orderTopic + "EMSX_DAY_AVG_PRICE,"


    subscriptions = blpapi.SubscriptionList()

    subscriptions.add(topic=orderTopic,correlationId=orderSubscriptionID)

    session.subscribe(subscriptions)
```

Pick and choose the elements and create route subscription:-

```
def createRouteSubscription(self, session):

    print "Create Route subscription"

    routeTopic = d_service + "/route?fields="
    routeTopic = routeTopic + "API_SEQ_NUM,"
```

(continues on next page)

(continued from previous page)

```
    routeTopic = routeTopic + "EMSX_AMOUNT,"
    routeTopic = routeTopic + "EMSX_AVG_PRICE,"
    routeTopic = routeTopic + "EMSX_BROKER,"
    routeTopic = routeTopic + "EMSX_BROKER_COMM,"
    routeTopic = routeTopic + "EMSX_BSE_AVG_PRICE,"
    routeTopic = routeTopic + "EMSX_BSE_FILLED,"
    routeTopic = routeTopic + "EMSX_CLEARING_ACCOUNT,"
    routeTopic = routeTopic + "EMSX_CLEARING_FIRM,"



    subscriptions = blpapi.SubscriptionList()

    subscriptions.add(topic=routeTopic,correlationId=routeSubscriptionID)

    session.subscribe(subscriptions)
```

Output:-

```
C:\Users\_scripts>py -3 EMSXSubscriptions_beta.py
Bloomberg - EMSX API Example - EMSXSubscriptions
Connecting to localhost:8194
Press ENTER to quit
Processing SESSION_STATUS event
Session connection is up
Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Create Order subscription
Processing SUBSCRIPTION_STATUS event
Order subscription started successfully
Create Route subscription

ORDER MESSAGE: CorrelationID(98)   Status(4)
MESSAGE: OrderRouteFields = {
        MSG_TYPE = "E"
        MSG_SUB_TYPE = "O"
        EMSX_SEQUENCE = 4747927
        EMSX_ROUTE_ID = 0
        EMSX_FILL_ID = 0
        API_SEQ_NUM = 1
        EVENT_STATUS = 4
        EMSX_ACCOUNT = ""
        EMSX_AMOUNT = 6000
        EMSX_ASSET_CLASS = "Equity"
        EMSX_ASSIGNED_TRADER = ""
        EMSX_AVG_PRICE = 161.330000
        EMSX_BASKET_NAME = ""
        EMSX_BASKET_NUM = 0
        EMSX_BLOCK_ID = ""
        EMSX_BROKER = ""
        EMSX_BROKER_COMM = 0.000000
        EMSX_BSE_AVG_PRICE = 0.000000
        EMSX_BSE_FILLED = 0
        EMSX_BUYSIDE_LEI = ""
        EMSX_CFD_FLAG = "N"
```

(continues on next page)

(continued from previous page)

```
        EMSX_CLIENT_IDENTIFICATION = ""
        EMSX_COMM_RATE = 0.000000
        EMSX_CURRENCY_PAIR = ""
        EMSX_DATE = 20200113
        EMSX_DAY_AVG_PRICE = 161.330000
        EMSX_DAY_FILL = 360
        EMSX_DIR_BROKER_FLAG = "N"
        EMSX_EXCHANGE = "US"
        EMSX_EXCHANGE_DESTINATION = "ANY"
        EMSX_EXEC_INSTRUCTION = ""
        EMSX_FILLED = 360
        EMSX_GPI = ""
        EMSX_GTD_DATE = 0
        EMSX_HAND_INSTRUCTION = "ANY"
        EMSX_IDLE_AMOUNT = 5580
        EMSX_INVESTOR_ID = "InvID"
        EMSX_ISIN = "US5949181045"
        EMSX_LIMIT_PRICE = 0.000000
        EMSX_MIFID_II_INSTRUCTION = ""
        EMSX_NOTES = ""
        EMSX_NSE_AVG_PRICE = 0.000000
        EMSX_NSE_FILLED = 0
        EMSX_ORD_REF_ID = ""
        EMSX_ORDER_AS_OF_DATE = 20200113
        EMSX_ORDER_AS_OF_TIME_MICROSEC = 49794.000000
        EMSX_ORDER_TYPE = "MKT"
        EMSX_PERCENT_REMAIN = 94.000000
        EMSX_PM_UUID = 6767714
        EMSX_PORT_MGR = "TKIM94"
        EMSX_PORT_NAME = ""
        EMSX_PORT_NUM = 9999
        EMSX_POSITION = ""
        EMSX_PRINCIPAL = 58078.800000
        EMSX_PRODUCT = "Equity"
        EMSX_QUEUED_DATE = 0
        EMSX_QUEUED_TIME = 0
        EMSX_QUEUED_TIME_MICROSEC = 0.000000
        EMSX_REASON_CODE = ""
        EMSX_REASON_DESC = ""
        EMSX_REMAIN_BALANCE = 5640.000000
        EMSX_ROUTE_PRICE = 0.000000
        EMSX_SEC_NAME = "MICROSOFT CORP"
        EMSX_SEDOL = "2588173 "
        EMSX_SETTLE_AMOUNT = 0.000000
        EMSX_SETTLE_DATE = 0
        EMSX_SI = "N"
        EMSX_SIDE = "BUY"
        EMSX_START_AMOUNT = 1100
        EMSX_STATUS = "WORKING"
        EMSX_STEP_OUT_BROKER = ""
        EMSX_STOP_PRICE = 0.000000
        EMSX_STRATEGY_END_TIME = 0
        EMSX_STRATEGY_PART_RATE1 = 0.000000
        EMSX_STRATEGY_PART_RATE2 = 0.000000
        EMSX_STRATEGY_STYLE = ""
        EMSX_STRATEGY_TYPE = ""
        EMSX_TICKER = "MSFT US Equity"
```

(continues on next page)

(continued from previous page)

```
        EMSX_TIF = "DAY"
        EMSX_TIME_STAMP = 49794
        EMSX_TIME_STAMP_MICROSEC = 49794.341000
        EMSX_TRAD_UUID = 6767714
        EMSX_TRADE_DESK = ""
        EMSX_TRADER = "TKIM94"
        EMSX_TRADER_NOTES = ""
        EMSX_TS_ORDNUM = -4747927
        EMSX_TYPE = "MKT"
        EMSX_UNDERLYING_TICKER = "Loading"
        EMSX_USER_COMM_AMOUNT = 0.000000
        EMSX_USER_COMM_RATE = 0.000000
        EMSX_USER_FEES = 0.000000
        EMSX_USER_NET_MONEY = 58078.800000
        EMSX_WORK_PRICE = 0.000000
        EMSX_WORKING = 60
        EMSX_YELLOW_KEY = "Equity"
        EMSX_STRATEGY_START_TIME = 0
        EMSX_CUSTOM_NOTE1 = ""
        EMSX_CUSTOM_NOTE2 = ""
        EMSX_CUSTOM_NOTE3 = ""
        EMSX_CUSTOM_NOTE4 = ""
        EMSX_CUSTOM_NOTE5 = ""
        EMSX_MOD_PEND_STATUS = ""
}

API_SEQ_NUM: 1
EMSX_ACCOUNT:
EMSX_AMOUNT: 6000
EMSX_ASSET_CLASS: Equity
EMSX_ASSIGNED_TRADER:
EMSX_AVG_PRICE: 161
EMSX_BASKET_NAME:
EMSX_BASKET_NUM: 0
EMSX_BLOCK_ID:
EMSX_BROKER:
EMSX_BROKER_COMM: 0
EMSX_BSE_AVG_PRICE: 0
EMSX_BSE_FILLED: 0
EMSX_BUYSIDE_LEI:
EMSX_CFD_FLAG: N
EMSX_CLIENT_IDENTIFICATION:
EMSX_COMM_DIFF_FLAG:
EMSX_COMM_RATE: 0
EMSX_CUSTOM_NOTE1:
EMSX_CUSTOM_NOTE2:
EMSX_CUSTOM_NOTE3:
EMSX_CUSTOM_NOTE4:
EMSX_CUSTOM_NOTE5:
EMSX_CURRENCY_PAIR:
EMSX_DATE: 20200113
EMSX_DAY_AVG_PRICE: 161
EMSX_DAY_FILL: 360
EMSX_DIR_BROKER_FLAG: N
EMSX_EXCHANGE: US
EMSX_EXCHANGE_DESTINATION: ANY
EMSX_EXEC_INSTRUCTION:
```

(continues on next page)

(continued from previous page)

```
EMSX_FILL_ID: 0
EMSX_FILLED: 360
EMSX_GPI:
EMSX_GTD_DATE: 0
EMSX_HAND_INSTRUCTION: ANY
EMSX_IDLE_AMOUNT: 5580
EMSX_INVESTOR_ID: InvID
EMSX_ISIN: US5949181045
EMSX_LIMIT_PRICE: 0.00000000
EMSX_MIFID_II_INSTRUCTION:
EMSX_MOD_PEND_STATUS:
EMSX_NOTES:
EMSX_NSE_AVG_PRICE: 0
EMSX_NSE_FILLED: 0
EMSX_ORD_REF_ID:
EMSX_ORDER_AS_OF_DATE: 20200113
EMSX_ORDER_AS_OF_TIME_MICROSEC: 49794.00000000
EMSX_ORDER_TYPE: MKT
EMSX_ORIGINATE_TRADER:
EMSX_ORIGINATE_TRADER_FIRM:
EMSX_PERCENT_REMAIN: 94
EMSX_PM_UUID: 6767714
EMSX_PORT_MGR: TKIM94
EMSX_PORT_NAME:
EMSX_PORT_NUM: 9999
EMSX_POSITION:
EMSX_PRINCIPAL: 58078
EMSX_PRODUCT: Equity
EMSX_QUEUED_DATE: 0
EMSX_QUEUED_TIME: 0
EMSX_QUEUED_TIME_MICROSEC: 0.00000000
EMSX_REASON_CODE:
EMSX_REASON_DESC:
EMSX_REMAIN_BALANCE: 5640
EMSX_ROUTE_ID: 0
EMSX_ROUTE_PRICE: 0
EMSX_SEC_NAME: MICROSOFT CORP
EMSX_SEDOL: 2588173
EMSX_SEQUENCE: 4747927
EMSX_SETTLE_AMOUNT: 0
EMSX_SETTLE_DATE: 0
EMSX_SI: N
EMSX_SIDE: BUY
EMSX_START_AMOUNT: 1100
EMSX_STATUS: WORKING
EMSX_STEP_OUT_BROKER:
EMSX_STOP_PRICE: 0
EMSX_STRATEGY_END_TIME: 0
EMSX_STRATEGY_PART_RATE1: 0
EMSX_STRATEGY_PART_RATE2: 0
EMSX_STRATEGY_STYLE:
EMSX_STRATEGY_TYPE:
EMSX_TICKER: MSFT US Equity
EMSX_TIF: DAY
EMSX_TIME_STAMP: 49794
EMSX_TIME_STAMP_MICROSEC: 49794.34100000
EMSX_TRAD_UUID: 6767714
```

(continues on next page)

(continued from previous page)

```
EMSX_TRADE_DESK:
EMSX_TRADER: TKIM94
EMSX_TRADER_NOTES:
EMSX_TS_ORDNUM: -4747927
EMSX_TYPE: MKT
EMSX_UNDERLYING_TICKER: Loading
EMSX_USER_COMM_AMOUNT: 0
EMSX_USER_COMM_RATE: 0
EMSX_USER_FEES: 0
EMSX_USER_NET_MONEY: 58078
EMSX_WORK_PRICE: 0
EMSX_WORKING: 60
EMSX_YELLOW_KEY: Equity
Processing SUBSCRIPTION_STATUS event
Route subscription started successfully

ORDER MESSAGE: CorrelationID(98)   Status(4)
MESSAGE: OrderRouteFields = {
        MSG_TYPE = "E"
        MSG_SUB_TYPE = "O"
        EMSX_SEQUENCE = 4747928
        EMSX_ROUTE_ID = 0
        EMSX_FILL_ID = 0
        API_SEQ_NUM = 2
        EVENT_STATUS = 4
        EMSX_ACCOUNT = ""
        EMSX_AMOUNT = 1100
        EMSX_ASSET_CLASS = "Equity"
        EMSX_ASSIGNED_TRADER = ""
        EMSX_AVG_PRICE = 161.330000
        EMSX_BASKET_NAME = ""
        EMSX_BASKET_NUM = 0
        EMSX_BLOCK_ID = ""
        EMSX_BROKER = ""
        EMSX_BROKER_COMM = 0.000000
        EMSX_BSE_AVG_PRICE = 0.000000
        EMSX_BSE_FILLED = 0
        EMSX_BUYSIDE_LEI = ""
        EMSX_CFD_FLAG = "N"
        EMSX_CLIENT_IDENTIFICATION = ""
        EMSX_COMM_RATE = 0.000000
        EMSX_CURRENCY_PAIR = ""
        EMSX_DATE = 20200113
        EMSX_DAY_AVG_PRICE = 161.330000
        EMSX_DAY_FILL = 198
        EMSX_DIR_BROKER_FLAG = "N"
        EMSX_EXCHANGE = "US"
        EMSX_EXCHANGE_DESTINATION = "ANY"
        EMSX_EXEC_INSTRUCTION = ""
        EMSX_FILLED = 198
        EMSX_GPI = ""
        EMSX_GTD_DATE = 0
        EMSX_HAND_INSTRUCTION = "ANY"
        EMSX_IDLE_AMOUNT = 600
        EMSX_INVESTOR_ID = ""
        EMSX_ISIN = "US5949181045"
        EMSX_LIMIT_PRICE = 0.000000
```

(continues on next page)

(continued from previous page)

```
        EMSX_MIFID_II_INSTRUCTION = ""
        EMSX_NOTES = ""
        EMSX_NSE_AVG_PRICE = 0.000000
        EMSX_NSE_FILLED = 0
        EMSX_ORD_REF_ID = ""
        EMSX_ORDER_AS_OF_DATE = 20200113
        EMSX_ORDER_AS_OF_TIME_MICROSEC = 49797.000000
        EMSX_ORDER_TYPE = "MKT"
        EMSX_PERCENT_REMAIN = 82.000000
        EMSX_PM_UUID = 6767714
        EMSX_PORT_MGR = "TKIM94"
        EMSX_PORT_NAME = ""
        EMSX_PORT_NUM = 9999
        EMSX_POSITION = ""
        EMSX_PRINCIPAL = 31943.340000
        EMSX_PRODUCT = "Equity"
        EMSX_QUEUED_DATE = 0
        EMSX_QUEUED_TIME = 0
        EMSX_QUEUED_TIME_MICROSEC = 0.000000
        EMSX_REASON_CODE = ""
        EMSX_REASON_DESC = ""
        EMSX_REMAIN_BALANCE = 902.000000
        EMSX_ROUTE_PRICE = 0.000000
        EMSX_SEC_NAME = "MICROSOFT CORP"
        EMSX_SEDOL = "2588173 "
        EMSX_SETTLE_AMOUNT = 0.000000
        EMSX_SETTLE_DATE = 0
        EMSX_SI = "N"
        EMSX_SIDE = "BUY"
        EMSX_START_AMOUNT = 1100
        EMSX_STATUS = "WORKING"
        EMSX_STEP_OUT_BROKER = ""
        EMSX_STOP_PRICE = 0.000000
        EMSX_STRATEGY_END_TIME = 0
        EMSX_STRATEGY_PART_RATE1 = 0.000000
        EMSX_STRATEGY_PART_RATE2 = 0.000000
        EMSX_STRATEGY_STYLE = ""
        EMSX_STRATEGY_TYPE = ""
        EMSX_TICKER = "MSFT US Equity"
        EMSX_TIF = "DAY"
        EMSX_TIME_STAMP = 49797
        EMSX_TIME_STAMP_MICROSEC = 49797.410000
        EMSX_TRAD_UUID = 6767714
        EMSX_TRADE_DESK = ""
        EMSX_TRADER = "TKIM94"
        EMSX_TRADER_NOTES = ""
        EMSX_TS_ORDNUM = -4747928
        EMSX_TYPE = "MKT"
        EMSX_UNDERLYING_TICKER = "Loading"
        EMSX_USER_COMM_AMOUNT = 0.000000
        EMSX_USER_COMM_RATE = 0.000000
        EMSX_USER_FEES = 0.000000
        EMSX_USER_NET_MONEY = 31943.340000
        EMSX_WORK_PRICE = 0.000000
        EMSX_WORKING = 302
        EMSX_YELLOW_KEY = "Equity"
        EMSX_STRATEGY_START_TIME = 0
```

(continues on next page)

(continued from previous page)

```
        EMSX_CUSTOM_NOTE1 = ""
        EMSX_CUSTOM_NOTE2 = ""
        EMSX_CUSTOM_NOTE3 = ""
        EMSX_CUSTOM_NOTE4 = ""
        EMSX_CUSTOM_NOTE5 = ""
        EMSX_MOD_PEND_STATUS = ""
}

API_SEQ_NUM: 2
EMSX_ACCOUNT:
EMSX_AMOUNT: 1100
EMSX_ASSET_CLASS: Equity
EMSX_ASSIGNED_TRADER:
EMSX_AVG_PRICE: 161
EMSX_BASKET_NAME:
EMSX_BASKET_NUM: 0
EMSX_BLOCK_ID:
EMSX_BROKER:
EMSX_BROKER_COMM: 0
EMSX_BSE_AVG_PRICE: 0
EMSX_BSE_FILLED: 0
EMSX_BUYSIDE_LEI:
EMSX_CFD_FLAG: N
EMSX_CLIENT_IDENTIFICATION:
EMSX_COMM_DIFF_FLAG:
EMSX_COMM_RATE: 0
EMSX_CUSTOM_NOTE1:
EMSX_CUSTOM_NOTE2:
EMSX_CUSTOM_NOTE3:
EMSX_CUSTOM_NOTE4:
EMSX_CUSTOM_NOTE5:
EMSX_CURRENCY_PAIR:
EMSX_DATE: 20200113
EMSX_DAY_AVG_PRICE: 161
EMSX_DAY_FILL: 198
EMSX_DIR_BROKER_FLAG: N
EMSX_EXCHANGE: US
EMSX_EXCHANGE_DESTINATION: ANY
EMSX_EXEC_INSTRUCTION:
EMSX_FILL_ID: 0
EMSX_FILLED: 198
EMSX_GPI:
EMSX_GTD_DATE: 0
EMSX_HAND_INSTRUCTION: ANY
EMSX_IDLE_AMOUNT: 600
EMSX_INVESTOR_ID:
EMSX_ISIN: US5949181045
EMSX_LIMIT_PRICE: 0.00000000
EMSX_MIFID_II_INSTRUCTION:
EMSX_MOD_PEND_STATUS:
EMSX_NOTES:
EMSX_NSE_AVG_PRICE: 0
EMSX_NSE_FILLED: 0
EMSX_ORD_REF_ID:
EMSX_ORDER_AS_OF_DATE: 20200113
EMSX_ORDER_AS_OF_TIME_MICROSEC: 49797.00000000
EMSX_ORDER_TYPE: MKT
```

(continues on next page)

(continued from previous page)

```
EMSX_ORIGINATE_TRADER:
EMSX_ORIGINATE_TRADER_FIRM:
EMSX_PERCENT_REMAIN: 82
EMSX_PM_UUID: 6767714
EMSX_PORT_MGR: TKIM94
EMSX_PORT_NAME:
EMSX_PORT_NUM: 9999
EMSX_POSITION:
EMSX_PRINCIPAL: 31943
EMSX_PRODUCT: Equity
EMSX_QUEUED_DATE: 0
EMSX_QUEUED_TIME: 0
EMSX_QUEUED_TIME_MICROSEC: 0.00000000
EMSX_REASON_CODE:
EMSX_REASON_DESC:
EMSX_REMAIN_BALANCE: 902
EMSX_ROUTE_ID: 0
EMSX_ROUTE_PRICE: 0
EMSX_SEC_NAME: MICROSOFT CORP
EMSX_SEDOL: 2588173
EMSX_SEQUENCE: 4747928
EMSX_SETTLE_AMOUNT: 0
EMSX_SETTLE_DATE: 0
EMSX_SI: N
EMSX_SIDE: BUY
EMSX_START_AMOUNT: 1100
EMSX_STATUS: WORKING
EMSX_STEP_OUT_BROKER:
EMSX_STOP_PRICE: 0
EMSX_STRATEGY_END_TIME: 0
EMSX_STRATEGY_PART_RATE1: 0
EMSX_STRATEGY_PART_RATE2: 0
EMSX_STRATEGY_STYLE:
EMSX_STRATEGY_TYPE:
EMSX_TICKER: MSFT US Equity
EMSX_TIF: DAY
EMSX_TIME_STAMP: 49797
EMSX_TIME_STAMP_MICROSEC: 49797.41000000
EMSX_TRAD_UUID: 6767714
EMSX_TRADE_DESK:
EMSX_TRADER: TKIM94
EMSX_TRADER_NOTES:
EMSX_TS_ORDNUM: -4747928
EMSX_TYPE: MKT
EMSX_UNDERLYING_TICKER: Loading
EMSX_USER_COMM_AMOUNT: 0
EMSX_USER_COMM_RATE: 0
EMSX_USER_FEES: 0
EMSX_USER_NET_MONEY: 31943
EMSX_WORK_PRICE: 0
EMSX_WORKING: 302
EMSX_YELLOW_KEY: Equity
Order - End of initial paint

ROUTE MESSAGE: CorrelationID(99)   Status(4)
MESSAGE: OrderRouteFields = {
        MSG_TYPE = "E"
```

(continues on next page)

(continued from previous page)

```
        MSG_SUB_TYPE = "R"
        EMSX_SEQUENCE = 4747928
        EMSX_ROUTE_ID = 1
        EMSX_FILL_ID = 13
        API_SEQ_NUM = 1
        EVENT_STATUS = 4
        EMSX_AMOUNT = 500
        EMSX_AVG_PRICE = 161.330000
        EMSX_BROKER = "BB"
        EMSX_BROKER_COMM = 0.000000
        EMSX_BSE_AVG_PRICE = 0.000000
        EMSX_BSE_FILLED = 0
        EMSX_BUYSIDE_LEI = ""
        EMSX_CLIENT_IDENTIFICATION = ""
        EMSX_COMM_RATE = 0.000000
        EMSX_CURRENCY_PAIR = ""
        EMSX_DAY_AVG_PRICE = 161.330000
        EMSX_DAY_FILL = 198
        EMSX_EXCHANGE_DESTINATION = "ANY"
        EMSX_EXEC_INSTRUCTION = ""
        EMSX_FILLED = 198
        EMSX_GPI = ""
        EMSX_GTD_DATE = 0
        EMSX_HAND_INSTRUCTION = "ANY"
        EMSX_LIMIT_PRICE = 0.000000
        EMSX_MIFID_II_INSTRUCTION = ""
        EMSX_NOTES = ""
        EMSX_NSE_AVG_PRICE = 0.000000
        EMSX_NSE_FILLED = 0
        EMSX_ORDER_TYPE = "MKT"
        EMSX_PERCENT_REMAIN = 60.400000
        EMSX_PRINCIPAL = 31943.340000
        EMSX_QUEUED_DATE = 0
        EMSX_QUEUED_TIME = 0
        EMSX_QUEUED_TIME_MICROSEC = 0.000000
        EMSX_REASON_CODE = ""
        EMSX_REASON_DESC = ""
        EMSX_REMAIN_BALANCE = 302.000000
        EMSX_ROUTE_PRICE = 162.835000
        EMSX_SETTLE_AMOUNT = 0.000000
        EMSX_SETTLE_DATE = 20200115
        EMSX_STATUS = "REPPEN"
        EMSX_STOP_PRICE = 0.000000
        EMSX_STRATEGY_END_TIME = 0
        EMSX_STRATEGY_PART_RATE1 = 0.000000
        EMSX_STRATEGY_PART_RATE2 = 0.000000
        EMSX_STRATEGY_STYLE = ""
        EMSX_STRATEGY_TYPE = ""
        EMSX_TIF = "DAY"
        EMSX_TIME_STAMP = 49904
        EMSX_TIME_STAMP_MICROSEC = 49904.123000
        EMSX_TYPE = "MKT"
        EMSX_USER_COMM_AMOUNT = 0.000000
        EMSX_USER_COMM_RATE = 0.000000
        EMSX_USER_FEES = 0.000000
        EMSX_USER_NET_MONEY = 31943.340000
        EMSX_WORKING = 302
```

(continues on next page)

(continued from previous page)

```
        EMSX_APA_MIC = ""
        EMSX_BROKER_LEI = ""
        EMSX_BROKER_SI = ""
        EMSX_BROKER_STATUS = ""
        EMSX_CLEARING_ACCOUNT = ""
        EMSX_CLEARING_FIRM = ""
        EMSX_CUSTOM_ACCOUNT = ""
        EMSX_EXECUTE_BROKER = ""
        EMSX_IS_MANUAL_ROUTE = 0
        EMSX_LAST_CAPACITY = ""
        EMSX_LAST_FILL_DATE = 20200113
        EMSX_LAST_FILL_TIME = 50074
        EMSX_LAST_FILL_TIME_MICROSEC = 50074.215000
        EMSX_LAST_MARKET = ""
        EMSX_LAST_PRICE = 0.000000
        EMSX_LAST_SHARES = 0
        EMSX_LEG_FILL_DATE_ADDED = 0
        EMSX_LEG_FILL_PRICE = 0.000000
        EMSX_LEG_FILL_SEQ_NO = 0
        EMSX_LEG_FILL_SHARES = 0.000000
        EMSX_LEG_FILL_SIDE = ""
        EMSX_LEG_FILL_TICKER = ""
        EMSX_MISC_FEES = 0.000000
        EMSX_ML_ID = ""
        EMSX_ML_LEG_QUANTITY = 500
        EMSX_ML_NUM_LEGS = 0
        EMSX_ML_PERCENT_FILLED = 39.600000
        EMSX_ML_RATIO = 0.000000
        EMSX_ML_REMAIN_BALANCE = -198.000000
        EMSX_ML_STRATEGY = ""
        EMSX_ML_TOTAL_QUANTITY = 0
        EMSX_OTC_FLAG = ""
        EMSX_P_A = ""
        EMSX_ROUTE_AS_OF_DATE = 20200113
        EMSX_ROUTE_AS_OF_TIME_MICROSEC = 49904.123000
        EMSX_ROUTE_CREATE_DATE = 20200113
        EMSX_ROUTE_CREATE_TIME = 49904
        EMSX_ROUTE_CREATE_TIME_MICROSEC = 49904.123000
        EMSX_ROUTE_LAST_UPDATE_TIME = 50083
        EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC = 50083.276000
        EMSX_ROUTE_REF_ID = "MyRouteRef2"
        EMSX_STRATEGY_START_TIME = 0
        EMSX_TRADE_REPORTING_INDICATOR = ""
        EMSX_TRANSACTION_REPORTING_MIC = ""
        EMSX_URGENCY_LEVEL = 0
        EMSX_WAIVER_FLAG = ""
}

API_SEQ_NUM: 1
EMSX_AMOUNT: 500
EMSX_APA_MIC:
EMSX_AVG_PRICE: 161
EMSX_BROKER: BB
EMSX_BROKER_COMM: 0
EMSX_BROKER_LEI:
EMSX_BROKER_SI:
EMSX_BROKER_STATUS:
```

(continues on next page)

(continued from previous page)

```
EMSX_BSE_AVG_PRICE: 0
EMSX_BSE_FILLED: 0
EMSX_BUYSIDE_LEI:
EMSX_CLEARING_ACCOUNT:
EMSX_CLEARING_FIRM:
EMSX_CLIENT_IDENTIFICATION:
EMSX_COMM_DIFF_FLAG:
EMSX_COMM_RATE: 0
EMSX_CURRENCY_PAIR:
EMSX_CUSTOM_ACCOUNT:
EMSX_DAY_AVG_PRICE: 161
EMSX_DAY_FILL: 198
EMSX_EXCHANGE_DESTINATION: ANY
EMSX_EXEC_INSTRUCTION:
EMSX_EXECUTE_BROKER:
EMSX_FILL_ID: 13
EMSX_FILLED: 198
EMSX_GPI:
EMSX_GTD_DATE: 0
EMSX_HAND_INSTRUCTION: ANY
EMSX_IS_MANUAL_ROUTE: 0
EMSX_LAST_CAPACITY:
EMSX_LAST_FILL_DATE: 20200113
EMSX_LAST_FILL_TIME: 50074
EMSX_LAST_FILL_TIME_MICROSEC: 50074.21500000
EMSX_LAST_MARKET:
EMSX_LAST_PRICE: 0
EMSX_LAST_SHARES: 0
EMSX_LEG_FILL_DATE_ADDED: 0
EMSX_LEG_FILL_PRICE: 0.00000000
EMSX_LEG_FILL_SEQ_NO: 0
EMSX_LEG_FILL_SHARES: 0.00000000
EMSX_LEG_FILL_SIDE:
EMSX_LEG_FILL_TICKER:
EMSX_LEG_FILL_TIME_ADDED: 0
EMSX_LIMIT_PRICE: 0.00000000
EMSX_MIFID_II_INSTRUCTION:
EMSX_MISC_FEES: 0
EMSX_ML_ID:
EMSX_ML_LEG_QUANTITY: 500
EMSX_ML_NUM_LEGS: 0
EMSX_ML_PERCENT_FILLED: 39
EMSX_ML_RATIO: 0
EMSX_ML_REMAIN_BALANCE: -198
EMSX_ML_STRATEGY:
EMSX_ML_TOTAL_QUANTITY: 0
EMSX_NOTES:
EMSX_NSE_AVG_PRICE: 0
EMSX_NSE_FILLED: 0
EMSX_ORDER_TYPE: MKT
EMSX_OTC_FLAG:
EMSX_P_A:
EMSX_PERCENT_REMAIN: 60
EMSX_PRINCIPAL: 31943
EMSX_QUEUED_DATE: 0
EMSX_QUEUED_TIME: 0
EMSX_QUEUED_TIME_MICROSEC: 0.00000000
```

(continues on next page)

(continued from previous page)

```
EMSX_REASON_CODE:
EMSX_REASON_DESC:
EMSX_REMAIN_BALANCE: 302
EMSX_ROUTE_AS_OF_DATE: 20200113
EMSX_ROUTE_AS_OF_TIME_MICROSEC: 49904.12300000
EMSX_ROUTE_CREATE_DATE: 20200113
EMSX_ROUTE_CREATE_TIME: 49904
EMSX_ROUTE_CREATE_TIME_MICROSEC: 49904.12300000
EMSX_ROUTE_ID: 1
EMSX_ROUTE_LAST_UPDATE_TIME: 50083
EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC: 50083.27600000
EMSX_ROUTE_PRICE: 162
EMSX_ROUTE_REF_ID: MyRouteRef2
EMSX_SEQUENCE: 4747928
EMSX_SETTLE_AMOUNT: 0
EMSX_SETTLE_DATE: 20200115
EMSX_STATUS: REPPEN
EMSX_STOP_PRICE: 0
EMSX_STRATEGY_END_TIME: 0
EMSX_STRATEGY_PART_RATE1: 0
EMSX_STRATEGY_PART_RATE2: 0
EMSX_STRATEGY_START_TIME: 0
EMSX_STRATEGY_STYLE:
EMSX_STRATEGY_TYPE:
EMSX_TIF: DAY
EMSX_TIME_STAMP: 49904
EMSX_TIME_STAMP_MICROSEC: 49904.12300000
EMSX_TRADE_REPORTING_INDICATOR:
EMSX_TRANSACTION_REPORTING_MIC:
EMSX_TYPE: MKT
EMSX_URGENCY_LEVEL: 0
EMSX_USER_COMM_AMOUNT: 0
EMSX_USER_COMM_RATE: 0
EMSX_USER_FEES: 0
EMSX_USER_NET_MONEY: 31943
EMSX_WAIVER_FLAG:
EMSX_WORKING: 302
EMSX_ROUTE_AS_OF_DATE: 20200113

ROUTE MESSAGE: CorrelationID(99)   Status(4)
MESSAGE: OrderRouteFields = {
        MSG_TYPE = "E"
        MSG_SUB_TYPE = "R"
        EMSX_SEQUENCE = 4747927
        EMSX_ROUTE_ID = 2
        EMSX_FILL_ID = 9
        API_SEQ_NUM = 2
        EVENT_STATUS = 4
        EMSX_AMOUNT = 200
        EMSX_AVG_PRICE = 161.330000
        EMSX_BROKER = "BB"
        EMSX_BROKER_COMM = 0.000000
        EMSX_BSE_AVG_PRICE = 0.000000
        EMSX_BSE_FILLED = 0
        EMSX_BUYSIDE_LEI = ""
        EMSX_CLIENT_IDENTIFICATION = ""
        EMSX_COMM_RATE = 0.000000
```

(continues on next page)

(continued from previous page)

```
        EMSX_CURRENCY_PAIR = ""
        EMSX_DAY_AVG_PRICE = 161.330000
        EMSX_DAY_FILL = 140
        EMSX_EXCHANGE_DESTINATION = "ANY"
        EMSX_EXEC_INSTRUCTION = "Work"
        EMSX_FILLED = 140
        EMSX_GPI = ""
        EMSX_GTD_DATE = 0
        EMSX_HAND_INSTRUCTION = "ANY"
        EMSX_LIMIT_PRICE = 0.000000
        EMSX_MIFID_II_INSTRUCTION = ""
        EMSX_NOTES = "Some notes"
        EMSX_NSE_AVG_PRICE = 0.000000
        EMSX_NSE_FILLED = 0
        EMSX_ORDER_TYPE = "MKT"
        EMSX_PERCENT_REMAIN = 30.000000
        EMSX_PRINCIPAL = 22586.200000
        EMSX_QUEUED_DATE = 0
        EMSX_QUEUED_TIME = 0
        EMSX_QUEUED_TIME_MICROSEC = 0.000000
        EMSX_REASON_CODE = ""
        EMSX_REASON_DESC = ""
        EMSX_REMAIN_BALANCE = 60.000000
        EMSX_ROUTE_PRICE = 162.785000
        EMSX_SETTLE_AMOUNT = 0.000000
        EMSX_SETTLE_DATE = 20200115
        EMSX_STATUS = "PARTFILL"
        EMSX_STOP_PRICE = 0.000000
        EMSX_STRATEGY_END_TIME = 0
        EMSX_STRATEGY_PART_RATE1 = 0.000000
        EMSX_STRATEGY_PART_RATE2 = 0.000000
        EMSX_STRATEGY_STYLE = ""
        EMSX_STRATEGY_TYPE = ""
        EMSX_TIF = "DAY"
        EMSX_TIME_STAMP = 50313
        EMSX_TIME_STAMP_MICROSEC = 50313.841000
        EMSX_TYPE = "MKT"
        EMSX_USER_COMM_AMOUNT = 0.000000
        EMSX_USER_COMM_RATE = 0.000000
        EMSX_USER_FEES = 0.000000
        EMSX_USER_NET_MONEY = 22586.200000
        EMSX_WORKING = 60
        EMSX_APA_MIC = ""
        EMSX_BROKER_LEI = ""
        EMSX_BROKER_SI = ""
        EMSX_BROKER_STATUS = ""
        EMSX_CLEARING_ACCOUNT = ""
        EMSX_CLEARING_FIRM = ""
        EMSX_CUSTOM_ACCOUNT = ""
        EMSX_EXECUTE_BROKER = ""
        EMSX_IS_MANUAL_ROUTE = 0
        EMSX_LAST_CAPACITY = ""
        EMSX_LAST_FILL_DATE = 20200113
        EMSX_LAST_FILL_TIME = 50443
        EMSX_LAST_FILL_TIME_MICROSEC = 50443.877000
        EMSX_LAST_MARKET = "N"
        EMSX_LAST_PRICE = 161.330000
```

(continues on next page)

(continued from previous page)

```
        EMSX_LAST_SHARES = 20
        EMSX_LEG_FILL_DATE_ADDED = 0
        EMSX_LEG_FILL_PRICE = 0.000000
        EMSX_LEG_FILL_SEQ_NO = 0
        EMSX_LEG_FILL_SHARES = 0.000000
        EMSX_LEG_FILL_SIDE = ""
        EMSX_LEG_FILL_TICKER = ""
        EMSX_MISC_FEES = 0.000000
        EMSX_ML_ID = ""
        EMSX_ML_LEG_QUANTITY = 200
        EMSX_ML_NUM_LEGS = 0
        EMSX_ML_PERCENT_FILLED = 70.000000
        EMSX_ML_RATIO = 0.000000
        EMSX_ML_REMAIN_BALANCE = -140.000000
        EMSX_ML_STRATEGY = ""
        EMSX_ML_TOTAL_QUANTITY = 0
        EMSX_OTC_FLAG = ""
        EMSX_P_A = ""
        EMSX_ROUTE_AS_OF_DATE = 20200113
        EMSX_ROUTE_AS_OF_TIME_MICROSEC = 50313.841000
        EMSX_ROUTE_CREATE_DATE = 20200113
        EMSX_ROUTE_CREATE_TIME = 50313
        EMSX_ROUTE_CREATE_TIME_MICROSEC = 50313.841000
        EMSX_ROUTE_LAST_UPDATE_TIME = 50443
        EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC = 50443.877000
        EMSX_ROUTE_REF_ID = ""
        EMSX_STRATEGY_START_TIME = 0
        EMSX_TRADE_REPORTING_INDICATOR = ""
        EMSX_TRANSACTION_REPORTING_MIC = ""
        EMSX_URGENCY_LEVEL = 0
        EMSX_WAIVER_FLAG = ""
}

API_SEQ_NUM: 2
EMSX_AMOUNT: 200
EMSX_APA_MIC:
EMSX_AVG_PRICE: 161
EMSX_BROKER: BB
EMSX_BROKER_COMM: 0
EMSX_BROKER_LEI:
EMSX_BROKER_SI:
EMSX_BROKER_STATUS:
EMSX_BSE_AVG_PRICE: 0
EMSX_BSE_FILLED: 0
EMSX_BUYSIDE_LEI:
EMSX_CLEARING_ACCOUNT:
EMSX_CLEARING_FIRM:
EMSX_CLIENT_IDENTIFICATION:
EMSX_COMM_DIFF_FLAG:
EMSX_COMM_RATE: 0
EMSX_CURRENCY_PAIR:
EMSX_CUSTOM_ACCOUNT:
EMSX_DAY_AVG_PRICE: 161
EMSX_DAY_FILL: 140
EMSX_EXCHANGE_DESTINATION: ANY
EMSX_EXEC_INSTRUCTION: Work
EMSX_EXECUTE_BROKER:
```

(continues on next page)

(continued from previous page)

```
EMSX_FILL_ID: 9
EMSX_FILLED: 140
EMSX_GPI:
EMSX_GTD_DATE: 0
EMSX_HAND_INSTRUCTION: ANY
EMSX_IS_MANUAL_ROUTE: 0
EMSX_LAST_CAPACITY:
EMSX_LAST_FILL_DATE: 20200113
EMSX_LAST_FILL_TIME: 50443
EMSX_LAST_FILL_TIME_MICROSEC: 50443.87700000
EMSX_LAST_MARKET: N
EMSX_LAST_PRICE: 161
EMSX_LAST_SHARES: 20
EMSX_LEG_FILL_DATE_ADDED: 0
EMSX_LEG_FILL_PRICE: 0.00000000
EMSX_LEG_FILL_SEQ_NO: 0
EMSX_LEG_FILL_SHARES: 0.00000000
EMSX_LEG_FILL_SIDE:
EMSX_LEG_FILL_TICKER:
EMSX_LEG_FILL_TIME_ADDED: 0
EMSX_LIMIT_PRICE: 0.00000000
EMSX_MIFID_II_INSTRUCTION:
EMSX_MISC_FEES: 0
EMSX_ML_ID:
EMSX_ML_LEG_QUANTITY: 200
EMSX_ML_NUM_LEGS: 0
EMSX_ML_PERCENT_FILLED: 70
EMSX_ML_RATIO: 0
EMSX_ML_REMAIN_BALANCE: -140
EMSX_ML_STRATEGY:
EMSX_ML_TOTAL_QUANTITY: 0
EMSX_NOTES: Some notes
EMSX_NSE_AVG_PRICE: 0
EMSX_NSE_FILLED: 0
EMSX_ORDER_TYPE: MKT
EMSX_OTC_FLAG:
EMSX_P_A:
EMSX_PERCENT_REMAIN: 30
EMSX_PRINCIPAL: 22586
EMSX_QUEUED_DATE: 0
EMSX_QUEUED_TIME: 0
EMSX_QUEUED_TIME_MICROSEC: 0.00000000
EMSX_REASON_CODE:
EMSX_REASON_DESC:
EMSX_REMAIN_BALANCE: 60
EMSX_ROUTE_AS_OF_DATE: 20200113
EMSX_ROUTE_AS_OF_TIME_MICROSEC: 50313.84100000
EMSX_ROUTE_CREATE_DATE: 20200113
EMSX_ROUTE_CREATE_TIME: 50313
EMSX_ROUTE_CREATE_TIME_MICROSEC: 50313.84100000
EMSX_ROUTE_ID: 2
EMSX_ROUTE_LAST_UPDATE_TIME: 50443
EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC: 50443.87700000
EMSX_ROUTE_PRICE: 162
EMSX_ROUTE_REF_ID:
EMSX_SEQUENCE: 4747927
EMSX_SETTLE_AMOUNT: 0
```

(continues on next page)

(continued from previous page)

```
EMSX_SETTLE_DATE: 20200115
EMSX_STATUS: PARTFILL
EMSX_STOP_PRICE: 0
EMSX_STRATEGY_END_TIME: 0
EMSX_STRATEGY_PART_RATE1: 0
EMSX_STRATEGY_PART_RATE2: 0
EMSX_STRATEGY_START_TIME: 0
EMSX_STRATEGY_STYLE:
EMSX_STRATEGY_TYPE:
EMSX_TIF: DAY
EMSX_TIME_STAMP: 50313
EMSX_TIME_STAMP_MICROSEC: 50313.84100000
EMSX_TRADE_REPORTING_INDICATOR:
EMSX_TRANSACTION_REPORTING_MIC:
EMSX_TYPE: MKT
EMSX_URGENCY_LEVEL: 0
EMSX_USER_COMM_AMOUNT: 0
EMSX_USER_COMM_RATE: 0
EMSX_USER_FEES: 0
EMSX_USER_NET_MONEY: 22586
EMSX_WAIVER_FLAG:
EMSX_WORKING: 60
EMSX_ROUTE_AS_OF_DATE: 20200113

ROUTE MESSAGE: CorrelationID(99)   Status(4)
MESSAGE: OrderRouteFields = {
        MSG_TYPE = "E"
        MSG_SUB_TYPE = "R"
        EMSX_SEQUENCE = 4747927
        EMSX_ROUTE_ID = 1
        EMSX_FILL_ID = 12
        API_SEQ_NUM = 3
        EVENT_STATUS = 4
        EMSX_AMOUNT = 220
        EMSX_AVG_PRICE = 161.330000
        EMSX_BROKER = "BB"
        EMSX_BROKER_COMM = 0.000000
        EMSX_BSE_AVG_PRICE = 0.000000
        EMSX_BSE_FILLED = 0
        EMSX_BUYSIDE_LEI = ""
        EMSX_CLIENT_IDENTIFICATION = ""
        EMSX_COMM_RATE = 0.000000
        EMSX_CURRENCY_PAIR = ""
        EMSX_DAY_AVG_PRICE = 161.330000
        EMSX_DAY_FILL = 220
        EMSX_EXCHANGE_DESTINATION = "ANY"
        EMSX_EXEC_INSTRUCTION = ""
        EMSX_FILLED = 220
        EMSX_GPI = ""
        EMSX_GTD_DATE = 0
        EMSX_HAND_INSTRUCTION = "ANY"
        EMSX_LIMIT_PRICE = 0.000000
        EMSX_MIFID_II_INSTRUCTION = ""
        EMSX_NOTES = ""
        EMSX_NSE_AVG_PRICE = 0.000000
        EMSX_NSE_FILLED = 0
        EMSX_ORDER_TYPE = "MKT"
```

(continues on next page)

(continued from previous page)

```
        EMSX_PERCENT_REMAIN = 0.000000
        EMSX_PRINCIPAL = 35492.600000
        EMSX_QUEUED_DATE = 0
        EMSX_QUEUED_TIME = 0
        EMSX_QUEUED_TIME_MICROSEC = 0.000000
        EMSX_REASON_CODE = ""
        EMSX_REASON_DESC = ""
        EMSX_REMAIN_BALANCE = 0.000000
        EMSX_ROUTE_PRICE = 162.835000
        EMSX_SETTLE_AMOUNT = 0.000000
        EMSX_SETTLE_DATE = 20200115
        EMSX_STATUS = "FILLED"
        EMSX_STOP_PRICE = 0.000000
        EMSX_STRATEGY_END_TIME = 0
        EMSX_STRATEGY_PART_RATE1 = 0.000000
        EMSX_STRATEGY_PART_RATE2 = 0.000000
        EMSX_STRATEGY_STYLE = ""
        EMSX_STRATEGY_TYPE = ""
        EMSX_TIF = "DAY"
        EMSX_TIME_STAMP = 49904
        EMSX_TIME_STAMP_MICROSEC = 49904.074000
        EMSX_TYPE = "MKT"
        EMSX_USER_COMM_AMOUNT = 0.000000
        EMSX_USER_COMM_RATE = 0.000000
        EMSX_USER_FEES = 0.000000
        EMSX_USER_NET_MONEY = 35492.600000
        EMSX_WORKING = 0
        EMSX_APA_MIC = ""
        EMSX_BROKER_LEI = ""
        EMSX_BROKER_SI = ""
        EMSX_BROKER_STATUS = ""
        EMSX_CLEARING_ACCOUNT = ""
        EMSX_CLEARING_FIRM = ""
        EMSX_CUSTOM_ACCOUNT = ""
        EMSX_EXECUTE_BROKER = ""
        EMSX_IS_MANUAL_ROUTE = 0
        EMSX_LAST_CAPACITY = ""
        EMSX_LAST_FILL_DATE = 20200113
        EMSX_LAST_FILL_TIME = 50104
        EMSX_LAST_FILL_TIME_MICROSEC = 50104.210000
        EMSX_LAST_MARKET = "N"
        EMSX_LAST_PRICE = 161.330000
        EMSX_LAST_SHARES = 22
        EMSX_LEG_FILL_DATE_ADDED = 0
        EMSX_LEG_FILL_PRICE = 0.000000
        EMSX_LEG_FILL_SEQ_NO = 0
        EMSX_LEG_FILL_SHARES = 0.000000
        EMSX_LEG_FILL_SIDE = ""
        EMSX_LEG_FILL_TICKER = ""
        EMSX_MISC_FEES = 0.000000
        EMSX_ML_ID = ""
        EMSX_ML_LEG_QUANTITY = 220
        EMSX_ML_NUM_LEGS = 0
        EMSX_ML_PERCENT_FILLED = 100.000000
        EMSX_ML_RATIO = 0.000000
        EMSX_ML_REMAIN_BALANCE = -220.000000
        EMSX_ML_STRATEGY = ""
```

(continues on next page)

(continued from previous page)

```
        EMSX_ML_TOTAL_QUANTITY = 0
        EMSX_OTC_FLAG = ""
        EMSX_P_A = ""
        EMSX_ROUTE_AS_OF_DATE = 20200113
        EMSX_ROUTE_AS_OF_TIME_MICROSEC = 49904.074000
        EMSX_ROUTE_CREATE_DATE = 20200113
        EMSX_ROUTE_CREATE_TIME = 49904
        EMSX_ROUTE_CREATE_TIME_MICROSEC = 49904.074000
        EMSX_ROUTE_LAST_UPDATE_TIME = 50104
        EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC = 50104.210000
        EMSX_ROUTE_REF_ID = "MyRouteRef1"
        EMSX_STRATEGY_START_TIME = 0
        EMSX_TRADE_REPORTING_INDICATOR = ""
        EMSX_TRANSACTION_REPORTING_MIC = ""
        EMSX_URGENCY_LEVEL = 0
        EMSX_WAIVER_FLAG = ""
}

API_SEQ_NUM: 3
EMSX_AMOUNT: 220
EMSX_APA_MIC:
EMSX_AVG_PRICE: 161
EMSX_BROKER: BB
EMSX_BROKER_COMM: 0
EMSX_BROKER_LEI:
EMSX_BROKER_SI:
EMSX_BROKER_STATUS:
EMSX_BSE_AVG_PRICE: 0
EMSX_BSE_FILLED: 0
EMSX_BUYSIDE_LEI:
EMSX_CLEARING_ACCOUNT:
EMSX_CLEARING_FIRM:
EMSX_CLIENT_IDENTIFICATION:
EMSX_COMM_DIFF_FLAG:
EMSX_COMM_RATE: 0
EMSX_CURRENCY_PAIR:
EMSX_CUSTOM_ACCOUNT:
EMSX_DAY_AVG_PRICE: 161
EMSX_DAY_FILL: 220
EMSX_EXCHANGE_DESTINATION: ANY
EMSX_EXEC_INSTRUCTION:
EMSX_EXECUTE_BROKER:
EMSX_FILL_ID: 12
EMSX_FILLED: 220
EMSX_GPI:
EMSX_GTD_DATE: 0
EMSX_HAND_INSTRUCTION: ANY
EMSX_IS_MANUAL_ROUTE: 0
EMSX_LAST_CAPACITY:
EMSX_LAST_FILL_DATE: 20200113
EMSX_LAST_FILL_TIME: 50104
EMSX_LAST_FILL_TIME_MICROSEC: 50104.21000000
EMSX_LAST_MARKET: N
EMSX_LAST_PRICE: 161
EMSX_LAST_SHARES: 22
EMSX_LEG_FILL_DATE_ADDED: 0
EMSX_LEG_FILL_PRICE: 0.00000000
```

(continues on next page)

(continued from previous page)

```
EMSX_LEG_FILL_SEQ_NO: 0
EMSX_LEG_FILL_SHARES: 0.00000000
EMSX_LEG_FILL_SIDE:
EMSX_LEG_FILL_TICKER:
EMSX_LEG_FILL_TIME_ADDED: 0
EMSX_LIMIT_PRICE: 0.00000000
EMSX_MIFID_II_INSTRUCTION:
EMSX_MISC_FEES: 0
EMSX_ML_ID:
EMSX_ML_LEG_QUANTITY: 220
EMSX_ML_NUM_LEGS: 0
EMSX_ML_PERCENT_FILLED: 100
EMSX_ML_RATIO: 0
EMSX_ML_REMAIN_BALANCE: -220
EMSX_ML_STRATEGY:
EMSX_ML_TOTAL_QUANTITY: 0
EMSX_NOTES:
EMSX_NSE_AVG_PRICE: 0
EMSX_NSE_FILLED: 0
EMSX_ORDER_TYPE: MKT
EMSX_OTC_FLAG:
EMSX_P_A:
EMSX_PERCENT_REMAIN: 0
EMSX_PRINCIPAL: 35492
EMSX_QUEUED_DATE: 0
EMSX_QUEUED_TIME: 0
EMSX_QUEUED_TIME_MICROSEC: 0.00000000
EMSX_REASON_CODE:
EMSX_REASON_DESC:
EMSX_REMAIN_BALANCE: 0
EMSX_ROUTE_AS_OF_DATE: 20200113
EMSX_ROUTE_AS_OF_TIME_MICROSEC: 49904.07400000
EMSX_ROUTE_CREATE_DATE: 20200113
EMSX_ROUTE_CREATE_TIME: 49904
EMSX_ROUTE_CREATE_TIME_MICROSEC: 49904.07400000
EMSX_ROUTE_ID: 1
EMSX_ROUTE_LAST_UPDATE_TIME: 50104
EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC: 50104.21000000
EMSX_ROUTE_PRICE: 162
EMSX_ROUTE_REF_ID: MyRouteRef1
EMSX_SEQUENCE: 4747927
EMSX_SETTLE_AMOUNT: 0
EMSX_SETTLE_DATE: 20200115
EMSX_STATUS: FILLED
EMSX_STOP_PRICE: 0
EMSX_STRATEGY_END_TIME: 0
EMSX_STRATEGY_PART_RATE1: 0
EMSX_STRATEGY_PART_RATE2: 0
EMSX_STRATEGY_START_TIME: 0
EMSX_STRATEGY_STYLE:
EMSX_STRATEGY_TYPE:
EMSX_TIF: DAY
EMSX_TIME_STAMP: 49904
EMSX_TIME_STAMP_MICROSEC: 49904.07400000
EMSX_TRADE_REPORTING_INDICATOR:
EMSX_TRANSACTION_REPORTING_MIC:
EMSX_TYPE: MKT
```

(continues on next page)

(continued from previous page)

```
EMSX_URGENCY_LEVEL: 0
EMSX_USER_COMM_AMOUNT: 0
EMSX_USER_COMM_RATE: 0
EMSX_USER_FEES: 0
EMSX_USER_NET_MONEY: 35492
EMSX_WAIVER_FLAG:
EMSX_WORKING: 0
EMSX_ROUTE_AS_OF_DATE: 20200113
Route - End of initial paint
O.
R.
O.
R.
Processing SESSION_STATUS event
Error: Session connection is down
Processing SESSION_STATUS event
Error: Session has been terminated
Ctrl+C pressed. Stopping...
```

3.18 EMSX History Request

EMSX history service provides individual fill information via request/response service. The service name is `//blp/emsx.history` for production and `//blp/emsx.history.uat` for test environment.

Important: This service should not be used as a replacement for route subscription service to capture fills information in real-time. Anyone found to abuse the service by making constant calls to the history service will be shutdown permanently by Bloomberg.

A UUID's fills are only available if any of the following criteria are met:

1. The user has at least one Export Fill profile in `EMSI<GO>`, or
2. The user belongs to a team that is setup for team fill export, or
3. The user is an EMSX API user and has EMSX API access turned on in `EMSS<GO>` setting.

Unlike the EMSX API service `//blp/emapisvc` and `//blp/emapisvc_beta`, the history service supports `PARTIAL_RESPONSE` events. The `PARTIAL_RESPONSE` event messages will return messages that are a subset of the information.

The EMSX history service goes back up to 30 days in history.

Note: Please note this service will not be available as part of `//blp/emapisvc` or `//blp/emapisvc_beta` service.

Unlike the `//blp/emapisvc` and `//blp/emapisvc_beta` service, `//blp/emsx.history` and `//blp/emsx.history.uat` service uses semi-camel character for the element names.

Important: Please note that the timezone of this service will always be in US EST timezone for the fills regardless of the TZDF setting for the UUID.

Please note that EMSX History should never be used as a replacement for route subscription for real-time fills and updates to routes.

Element	Description
Account	Trading account used in EMSX<GO>
Amount	Total quantity of the order
AssetClass	Asset class of the order
BasketId	ID of the basket
BasketName	Name of the basket
BBGID	BBGID field
BlockId	Block ID
Broker	Executing broker name
BrokerExceId	Broker Execution ID
BrokerOrderId	Broker Order ID
ClearingAccount	Clearing account detail
ClearingFirm	Clearing firm detail
ContractExpDate	Contract expiration date
CorrectedFillId	Corrected fill ID
Currency	Currency
Cusip	CUSIP
DateTimeOfFill	Date and time of the fill
Exchange	Exchange details
ExecPrevSeqNo	Previous sequence number of execution
ExecType	Execution type details (FILL,CANCEL,CORRECT and DFD)
ExecutingBroker	Executing broker details
FillId	ID of the fill
FillPrice	Price of the fill
FillShares	Number of share of the fill
InvestorID	Investor ID detail
IsCfd	CFD flag
Isin	ISIN detail
IsLeg	Is leg
LastCapacity	Last capacity field in EMSX<GO>
LastMarket	Last market detail
LimitPrice	Limit price detail
Liquidity	Last liquidity indicator 1,2,3,M,T,A [definition].
LocalExchangeSymbol	Local exchange symbol
LocateBroker	Locate broker detail
LocateId	Locate ID
LocateRequired	Flag to indicate whether or not short locate is required
MifidAggrFlag	Aggregation flag for MiFID II
MifidBuysideLei	Legal entity identifier in MiFID II for the buy-side
MifidGpi	Global personal identifier in MiFID II
MifidIsSi	Flag to indiciate systematic internalizer in MiFID II
MifidSellsideApaMic	Sell-side Approved Publication Arrangment (APA) Market Identifier Code (MIC)
MifidSellsideLei	Legal entity identifier in MiFID II for the sell-side
MifidSellsideOtcFlag	Sell-side OTC flag
MifidSellsideSiMic	Sell-side systematic internalizer MIC
MifidSellsideTri	
MifidSellsideTriMic	

Continued on next page

Table 2 – continued from previous page

Element	Description
MifidSellsideWaiverFlag	Sell-side waiver flag for MiFID II
MifidTradeInstr	Trade instruction for MiFID II
Mpid	
MultilegId	Multileg ID
NyOrderCreateAsOfDateTime	NY order create as of datetime
NyTranCreateAsOfDateTime	NY transaction create as of datetime
OCCSymbol	OCC symbol
OrderExecutionInstruction	Order execution instruction detail
OrderHandlingInstruction	Order handling instruction detail
OrderId	Order ID
OrderInstruction	Order instruction detail
OrderOrigin	Order origin detail
OrderReferenceId	Order reference ID detail
OriginatingTraderUuid	UUID of the originating trader
ReroutedBroker	Rerouted broker details
RouteCommissionAmount	Commission amount of the route
RouteCommissionRate	Commission rate of the route
RouteExecutionInstruction	Route execution instruction
RouteHandlingInstruction	Route handling instruction
RouteId	Route ID
RouteNetMoney	Route net money
RouteNotes	Route instructions
RouteShares	Route shares
SecurityName	Security name detail
Sedol	SEDOL
SettlementDate	Settlement date detail
Side	Side
StopPrice	Stop Price
StrategyType	Strategy Type
Ticker	Ticker
TIF	Time in Force
TraderName	Name of the trader
TraderUuid	Bloomberg UUID of the trader
Type	Order type
UserCommissionAmount	User commission amount
UserCommissionRate	User commission rate
UserFees	User fee detail
UserNetMoney	User net money detail
YellowKey	Bloomberg yellow key field detail

Full code sample:-

EMSX History cpp	EMSX History cs	EMSX History vba
EMSX History java	EMSX History py	

Hint: Please right click on the top code sample link to open in a new tab.

Specify service name and host/port :-

```
d_service="//blp/emsx.history.uat"
d_host="localhost"
d_port=8194
bEnd=False
```

Connect and create a session object:-

```
class SessionEventHandler():

    def processEvent(self, event, session):
        try:
            if event.eventType() == blpapi.Event.SESSION_STATUS:
                self.processSessionStatusEvent(event,session)

            elif event.eventType() == blpapi.Event.SERVICE_STATUS:
                self.processServiceStatusEvent(event,session)

            elif event.eventType() == blpapi.Event.RESPONSE or event.eventType() ==␣
↪blpapi.Event.PARTIAL_RESPONSE:
                self.processResponseEvent(event)

            else:
                self.processMiscEvents(event)

        except blpapi.Exception as e:
            print "Exception:  %s" % e.description()
        return False
```

Set elements (e.g. UUID, team name, and Date/Time range):-

```
service = session.getService(d_service)

request = service.createRequest("GetFills")

request.set("FromDateTime", "2017-02-08T00:00:00.000+00:00")
request.set("ToDateTime", "2017-02-11T23:59:00.000+00:00")

scope = request.getElement("Scope")

#scope.setChoice("Team")
#scope.setChoice("TradingSystem")
scope.setChoice("Uuids")

#scope.setElement("Team", "TEAM1")
#scope.setElement("TradingSystem",false)

scope.getElement("Uuids").appendValue(8049857)

'''
scope.getElement("Uuids").appendValue(14348220);
scope.getElement("Uuids").appendValue(8639067);
scope.getElement("Uuids").appendValue(4674574);
'''
```

Process response events:-

```
def processResponseEvent(self, event):
```

(continues on next page)

(continued from previous page)

```
    print "Processing RESPONSE event"

    for msg in event:

        if msg.correlationIds()[0].value() == self.requestID.value():
            print "MESSAGE TYPE: %s" % msg.messageType()

            if msg.messageType() == ERROR_INFO:
                errorCode = msg.getElementAsInteger("ERROR_CODE")
                errorMessage = msg.getElementAsString("ERROR_MESSAGE")
                print "ERROR CODE: %d\tERROR MESSAGE: %s" % (errorCode,errorMessage)
            elif msg.messageType() == GET_FILLS_RESPONSE:
```

Output:-

```
C:\Users\_scripts>py -3 EMSXHistory.py
Bloomberg - EMSX API Example - EMSXHistory
Connecting to localhost:8194
Processing SESSION_STATUS event
SessionConnectionUp = {
        server = "localhost:8194"
        encryptionStatus = "Clear"
}

Processing SESSION_STATUS event
Session started...
Processing SERVICE_STATUS event
Service opened...
Request: GetFills = {
        FromDateTime = 2019-10-01T00:00:00.000+00:00
        ToDateTime = 2020-01-13T23:59:00.000+00:00
        Scope = {
                Uuids[] = {
                        12345678
                }
        }
}

Processing RESPONSE event
MESSAGE TYPE: GetFillsResponse
Date: 2019-12-12T11:35:02.674-05:00
Fill ID: 3      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 10112
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T11:50:02.717-05:00
Fill ID: 4      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 20224
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T12:05:02.758-05:00
```

(continues on next page)

(continued from previous page)

```
Fill ID: 5      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 30336
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T12:20:02.799-05:00
Fill ID: 6      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 40448
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T12:35:02.841-05:00
Fill ID: 7      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 50560
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T12:50:02.881-05:00
Fill ID: 8      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 60672
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2019-12-12T13:05:02.923-05:00
Fill ID: 9      OrderId: 4733965        RouteId: 1
Ticker: FB      Asset Class: Equity     Yellow Key: Equity
Shares: 50      Price: 202.240000       Broker: BMTB    CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 70784
Basket ID: 0    Currency: USD   Multileg ID:
Account: testAccount    LocateId:                       LocateBroker: False
↪OCCSymbol:
Date: 2020-01-13T14:01:23.880-05:00
Fill ID: 11     OrderId: 4747927        RouteId: 2
Ticker: MSFT    Asset Class: Equity     Yellow Key: Equity
Shares: 20      Price: 161.330000       Broker: BB      CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 29039
Basket ID: 0    Currency: USD   Multileg ID:
Account:        LocateId:                       LocateBroker: False     OCCSymbol:
Date: 2020-01-13T14:01:53.882-05:00
Fill ID: 12     OrderId: 4747927        RouteId: 2
Ticker: MSFT    Asset Class: Equity     Yellow Key: Equity
Shares: 20      Price: 161.330000       Broker: BB      CFD: False
Commission: 0   Commission Rate: 0      Fees: 0 Net Money: 32266
Basket ID: 0    Currency: USD   Multileg ID:
Account:        LocateId:                       LocateBroker: False     OCCSymbol:
Processing SESSION_STATUS event
SessionConnectionDown = {
        server = "localhost:8194"
}
```

(continues on next page)

(continued from previous page)

```
Processing SESSION_STATUS event
SessionTerminated = {
}
```

CHAPTER 4

MiFID II

In preparation for the MiFID II, Bloomberg EMSX API is enhancing its workflow to provide clients with the needed data and solutions to fulfill MiFID II obligations (Trade reporting, Transaction reporting, best execution and order record keeping). Bloomberg EMSX API as part of Bloomberg EMSX will connect to Bloomberg RHUB to support client's regulatory obligations through a centralized access for ARM, APA, ORK (Bvault) and best execution (BTCA).

MiFID II Field Names	Type
EMSX_BUYSIDE_LEI	String
EMSX_BROKER_LEI	String
EMSX_TRADE_REPORTING_INDICATOR	String
EMSX_TRANSACTION_REPORTING_MIC	String
EMSX_APA_MIC	String
EMSX_OTC_FLAG	String
EMSX_WAIVER_FLAG	String
EMSX_LAST_CAPACITY	String
EMSX_CLIENT_IDENTIFICATION	String
EMSX_SI	Bool
EMSX_MIFID_II_INSTRUCTION	String
EMSX_BROKER_SI	String
EMSX_ROUTE_CREATE_TIME_MICROSEC	Float64
EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC	Float64
EMSX_TIME_STAMP_MICROSEC	Float64
EMSX_QUEUED_TIME_MICROSEC	Float64
EMSX_LAST_FILL_TIME_MICROSEC	Float64
EMSX_GPI	String

New time stamp elements:-

EMSX_ROUTE_CREATE_TIME_MICROSEC	EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC
EMSX_TIME_STAMP_MICROSEC	EMSX_QUEUED_TIME_MICROSEC
EMSX_LAST_FILL_TIME_MICROSEC	

Important: The new timestamps for the new elements are only microseconds if they extend out to full digits. (e.g. 0.000001)

Please note that all microsecond timestamp is in float64 type, [second].[microsecond] format.

From time to time they will be printed to the millisecond precision in cases when the microsecond timestamp from the back-end is not available. (e.g. 0.001)

New requests with the MiFID II elements:-

CreateOrder	CreateOrderAndRouteEx
GroupRouteRequestEx	ManualFill
ManualRouteEx	ModifyOrderRequestEx
ModifyTheRouteRequestEx	RouteOrderRequestsEx

Order subscription with the MiFID II elements:-

EMSX_BUYSIDE_LEI	EMSX_CLIENT_IDENTIFICATION
EMSX_GPI	EMSX_MIFID_II_INSTRUCTION
EMSX_QUEUED_TIME_MICROSEC	EMSX_SI
EMSX_TIME_STAMP_MICROSEC	

Route subscription with the MiFID II elements:-

EMSX_APA_MIC	EMSX_BUYSIDE_LEI
EMSX_BROKER_LEI	EMSX_BROKER_SI
EMSX_CLIENT_IDENTIFICATION	EMSX_GPI
EMSX_LAST_CAPACITY	EMSX_LAST_FILL_TIME_MICROSEC
EMSX_MIFID_II_INSTRUCTION	EMSX_OTC_FLAG
EMSX_QUEUED_TIME_MICROSEC	EMSX_ROUTE_CREATE_TIME_MICROSEC
EMSX_ROUTE_LAST_UPDATE_TIME_MICROSEC	EMSX_TRADE_REPORTING_INDICATOR
EMSX_TRANSACTION_REPORTING_MIC	EMSX_TIME_STAMP_MICROSEC
EMSX_WAIVER_FLAG	

Note: MiFID II acrynom definitions can be found in Glossary section of the document.

CHAPTER 5

IOI API Service

The IOI API Server provides Bloomberg users with the ability to both publish and listen to the indication of interest messages. (IOI)

The IOI API Server allows sell-sides to publish their IOI messages into Bloomberg and for buy-sides and others to subscribe to the IOI messages from Bloomberg.

The IOIs consists of Equities and Derivatives and this utilizes the same Bloomberg API 3.0 as EMSX API to automate the publishing and subscribing the indication of interest messages.

More details can be found on the following URL:

http://ioi-api-doc.readthedocs.io/en/latest/

CHAPTER 6

FAQ

6.1 General FAQ

- **What is a session?** Sessions are logical data stream connections and the EMSX API supports failover betweeen physical connections. During this failover, EMSX API will handle re-subscriptions for the end application.

  If you are using multiple bloomberg API services, it is recommended to use separate sessions to avoid delaying a fast stream with slow one. For most design, it's best to have separte session for real-time data vs. EMSX API or reference data service.

- **Should I open and close sessions as needed?** No, typically opening and closing a session is expensive for both the client's application and for Bloomberg back-end and thus unnecessary for most application designs while using EMSX API.

- **How do I specify a ticker?** The EMSX_TICKER field should be specified either as a FIGI, or as a full parsekeyable value, including security, exchange and asset class, e.g.: "IBM US Equity". Failure to provide an explicit value can lead to unpredictable behaviour.

- **Why can I not subscribe using ticker and fields like other APIs?** The EMSX service only allows users to subscribe to their own Orders and Routes (placements). Most applications will use only two subscriptions, one for Orders and one for Routes (placements). A list of EMSX fields is required when creating the subscriptions.

- **Why can't I see my orders and or routes in EMSX?** The most common cause is that the user is connected to the BETA machines on the API side, whilst using the PROD machine on the terminal. Switching one of these will normally resolve the problem.

- **What happens when I subscribe to route level element on the topic string of my order subscription and vice versa?** Your subscription will fail and will generate error similar to the following:

```
reason = {
        errorCode = 3
        description = "Invalid field passed in: Field=|EMSX_MOD_PEND_STATUS|"
        category = "-13"
}
```

- **How do I connect to the BETA machine of the terminal?** Use the function DGRT Y087<GO> on the terminal, followed by EMSX<GO>. This will connect that terminal window to the EMSX BETA machine. Please note that this only applies to that particular terminal window only. To return to PROD system on the terminal, type DGRT OFF<GO>
- **How do I connect to PROD or BETA in the API?** Two separate services are provided. These are `//blp/emapisvc` (PROD) and `//blp/emapisvc_beta` (TEST)
- **How do I match my requests to responses?** This is done in the same was as for other Bloomberg API services, with the use of `CorrelationID`.
- **What broker or simulator do I use?** When first enabled for BETA access, client will generally be enabled for BMTB or other internal Bloomberg simulator codes. A new development broker has recently been added called the API. To be enabled for other brokers in the LIVE environment, clients should contact the EMSX Help Desk.
- **How do I test my application with these simulators?** Test brokers (BB, BMTB, EFIX and API) are automated systems that respond a request in a predetermined way, based on the specified security in the request. Each test broker has a set of documented behaviors that clients can take advantage of to create test cases. These documents are currently provided on request.
- **Why am I not seeing events that affect my Routes?** This is normally caused by only having a subscription for Orders. A separate subscription is needed for route messages when using our programmable interface.
- **Why am I still seeing orders that I deleted or have completed?** Orders that were manually deleted, or completed in a previous session, will continue to transmit on the order. Check the `EMSX_STATUS` of the returned message to confirm if this is a live order. These orders will cease to report between 24 and 48 hours after they are deleted depending on the nature of the order.
- **Why is the value of a field returned as blank / zero?** This normally means that the user has not subscribed to that field in the original subscription. This can also mean that the user did not subscribe to the filed in the first place or is requesting for a static field.
- **Why is a field not being returned?** Some fields are specific to either Orders or Routes. You cannot subscribe to an Order field in the Route subscription and vice versa.

  The type of message will also dictate which fields will be returned. For `NEW_ORDER_ROUTE` and `INIT_PAINT` messages, all fields will be returned. However, for `UPD_ORDER_ROUTE`, the user will only receive a small number of static fields along with all those fields deemed to be 'dynamic', meaning they can change during the lifetime of the order or route.

  This is one of the reasons as why the user is encouraged to maintain their own image of and order or route within their application.
- **How do I receive Fill messages?** Currently, the easiest way to track individual fills is to use the `//blp/emsx.history` service using request/response service calls.

  However, please do not use this as a replacement for the route subscription. Anyone constantly calling the history service and abusing the history service will be shut down by Bloomberg.

  The other option is to use the route subscription service. Each individual fill events will generate a `UPD_ORDER_ROUTE` the message, with the applicable changes to the order and route data.
- **I do not see the fill information for one of my team member when I call the history service using team name.** A UUID's fills are only stored if any of the following criteria are met:

  1. The user has at least one Export Fill profile in `EMSI<GO>`, or
  2. The user belongs to a team that is setup for team fill export, or
  3. The user is an EMSX API user, i.e., `EMSS<GO>` internal settings show "Enable EMSX API" to be true.

If the above criteria are not met, there will be no fills data history service can call to export.

- **How do I route a complete basket?** The term basket here is defined as a way to send the entire group of order into a single basket to a broker destination or to a broker algorithm, which supports basket. The term basket here is not intended for those who want to tie a particular group of orders into a trading strategy.

  Currently routing a basket is a two-step process in EMSX API. First, the user will need to use `CreateOrder` request to create the order and include the `EMSX_BASKET_NAME` in the field. To route the order, the user can use either `GroupRouteEx` or `GroupRouteWithStrat` and include the `EMSX_SEQUENCE` number inside the array.

  If the user misses an `EMSX_SEQUENCE` number inside the specified basket, the particular missing order will not be sent as part of the basket. This is the same logic used on EMSX<GO> for basket creation and basket submission.

- **How long do DAY orders and complete orders stay on the blotter and in the API?** (Status = 8)** In the old logic, the DAY orders stayed 4 hours after the exchange closed. The new logic is to extend this to 8 hours after the exchange closed. Expired orders are deleted after 2 days. For expired orders, when user gets `INIT_PAINT`, the will get updates for those expired orders with status=8.

  For partially filled orders delete will modify amount down to the filled amount and that order will not disappear and will be treated as a filled order. The Excel Add-In currently removes anything in the blotter with Status=8.

- **Why do I get "Internal error. Please contact customer support"?** Unfortunately, this is a generic error message, which can be caused by a number of reasons. However, the most common is that the user has failed to provide a mandatory field with a request.

- **Why do I get "Customer ABCDE is not validated for ETORSA"?** Client must sign a Bloomberg Electronic Trading & Order Routing Service Agreement before they can be enabled for EMSX API access.

- **Why do I get "User ABCDE is not permitted for the API"?** EMSX Help Desk must enable users for EMSX API access via EMSS.

- **Why do I get "User NOT Enabled to route to this broker by EOR (ENAB)."?** Users must be enabled for specific brokers. This is done by EMSX Help Desk support for internal simulator codes and by the broker for their own production codes.

- I am enabled but I get a red bar on the bottom when I click on the EMSX button.

  This is usually due to the following issues.

  - BBCOMM failed to establish a session. For this please see the next section on restarting BBCOMM
  - The ETORSA/FIET paperwork is not in file. Every EMSX API user's firm will need to sign ETORSA and or FIET before using the EMSX API. Please click Help Help in EMSX<GO> and have the Trade Desk personnel check for this legal check.
  - The desktop prevents any third party WPF components from running. This is usually tied into the PC's image. This will usually cause an exception in the System.Windows.Media.Composition library. This will usually require reinstall of .NET 3.5 SP1, hardware display drivers, and DirectX libraries.

- How do I restart bbcomm?
  - Close all instances of Excel, Word and PowerPoint.
  - Open task manager and kill bxlaui.exe and bxlartd.exe.
  - Open a command prompt and type bbstop

  - In the same command prompt, type the command bbcomm. BBCOMM should report that it is running successfully and should not return.

- How do I regenerate apiregistry.ini file?

  Open regedit from RUN window and Clear the "APIRegistryCRC32" registry value located at "HKEY_LOCAL_MACHINESOFTWAREBloomberg L.P.Office ToolsSettings" or "HKEY_LOCAL_MACHINESOFTWAREWow6432NodeBloomberg L.P.Office ToolsSettings" on Windows 7.

- **How do I modify GTD to day order?** Set `EMSX_GTD_DATE` to "-1" or -1 or any negative GTD date will reset the order to day order.
- **How do I modify or reset the stop price of an order?** Set `EMSX_STOP_PRICE` to "-1" or -1
- **How do I reset my order from Limit to Market?** `EMSX_LIMIT_PRICE = -99999` is only required when modifying *from* LMT to something else
- **How do I set 0 limit price for futures spread orders?** `EMSX_LIMIT_PRICE = -99999` needs to be set, otherwise the 0 limit price will be ignored.
- **How is `EMSX_RELEASE_TIME` used?** `EMSX_RELEASE_TIME` is in `HH:MM` format. For the API it is defaulted to the exchange time. This only works on requests that are routable from EMSX API. Thus, it will not work on CreateOrder request. Since the field is an integer, it should be forammted as `1101` for `11:01`.
- **Are `EMSX_TICKER` and `EMSX_SIDE` elements always available on the subscription service?** No, any fields that are static are not always returned.
- **Can update events come before the `INITIAL_PAINT` or new event?** Yes, this wasn't the original intention, however, due to current EMSX back-end, the update `Event Status = 7` messages can come before `INITIAL_PAINT` `Event Status=4` or New `Event Status = 6`
- **Are `INITIAL_PAINT` messages always first?** No, you can receive any route messages before the order message with `INITIAL_PAINT`.
- **Is there any downtime for EMSX API service?** Yes, generally for EMSX services, it is down during machin maintenance on Saturday from 1pm to 5pm Eastern Standard Time. For API routers, the routers are turned from Sunday US between 9am-1pm US Eastern Standard Time. During the weekend turnaround, services are down during this time time and there will be no access to the service. The dependencies here are on the the machines the services resides and not the service itself.
- **Is there a community project based on EMSX API?** Yes, there is a MIT licensed community project. It's called EasyMSX.

CHAPTER 7

Glossary

AIM Bloomberg Asset and Investment Manager (OMS).

APA Approved Publication Arrangement in MiFID II.

API Application Programming Interface. The definition of the way in which two applicaitons can communciate with each other.

Authentication The act of identifying and ahtorizing the user when creating their identity.

BAMS The Bloomberg Appliance Management System that enables the real-time monitoring of Bloomberg appliances (servers).

Bloomberg API The Application Programming Interface (API) provided by Bloomberg that allows developers to access data services from within custom built appliations written in C, C++, Java, .Net languages (C#, VB and etc.), Python and Excel VBA

Bloomberg App Portal The storefront within the Bloomberg terminal that allows clients to download applications to run within the Bloomberg launchpad.

COM Control Microsoft Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. For EMSX API, this is a style of library used for Excel.

EMS A generic term for execution management system.

E2E EMSX-to-EMSX. A specific arrangement where both the buy-side and the sell-side are using EMSX<GO>.

EMSX<GO> Bloomberg Execution Management System (EMS) for equities, futures and options.

EMSxNET FIX network offering from Bloomberg that allows clients to transmit data in the FIX protocol across Bloomberg network infrastructure.

GPI Global Personal Identifier in MiFID II.

LEI Legal Entity Identifier in MiFID II.

LMNU terminal The limited functionality terminal. There are temporary terminals provided to clients for specific task. For EMSX API, the LMNU needed is 20025.

Market Data The market data service of Bloomberg API: `//blp/mktdata`

MIC Market Identifier Code.

Non-BPS Any user who does not use a Bloomberg terminal, but has access to the Bloomberg API.

OMS A generic term for order management system.

Placement Creating a route in EMSX<GO> is essentially a buy-side placement to the market. Going forward we will refer all placement as routes in the documentation.

Reference Data The reference data service of the Bloomberg API: `//blp/refdata`

Route Technically route refers to orders being submitted from the execution brokers to exchanges. EMSX<GO> uses the term route to mean placement.

Server-Side Application Server-side refers to operations that are performed by the server and not by the desktop such as PC.

Service Refers to the different types of data connections available via the Bloomberg API. Each service has its own schema that describes what can be done, and what data fields are available to the application, For example, market data, reference data, EMSX API, and etc.

SI Systematic Internalizer in MiFID II.

SLA Service level agreement

WAPI<GO> The Bloomberg terminal function where the user can download the SDK for Bloomberg API.

Bibliography

[definition] 1=Added, 2=Removed, 3=Routed out, M=Maker, T=Taker, R=Rerouted, A=Auction