

EMSX API

Programmer's Guide

Excel Add-In & Programmable

Version 4.3

This Bloomberg EMSX API documentation is a reference guide for Bloomberg customers and authorized developers, and is provided for such informational purposes only. The documentation cannot be released to anyone who is not licensed to receive it from Bloomberg. The documentation is governed by the Bloomberg Agreement and other agreements with Bloomberg. BLOOMBERG is a registered trademark of Bloomberg, L.P. in the United States and other countries. All other trademarks and registered trademarks are the property of their respective owners. © 2011 Bloomberg, L.P. All rights reserved.

FRANKFURT	HONG KONG	LONDON	NEW YORK	SAN FRANCISCO	SÃO PAULO	SINGAPORE	SYDNEY	TOKYO
+49 69 9204 1210	+852 2977 6000	+44 20 7330 7500	+1 212 318 2000	+1 415 912 2960	+55 11 3048 4500	+65 6212 1000	+612 9777 8600	+81 3 3201 8900



Press the <HELP>
key twice for instant
live assistance.

The BLOOMBERG PROFESSIONAL service, BLOOMBERG Data and BLOOMBERG Order Management Systems (the "Services") are owned and distributed locally by Bloomberg Finance L.P. ("BFLP") and its subsidiaries in all jurisdictions other than Argentina, Bermuda, China, India, Japan and Korea (the "BLP Countries"). BFLP is a wholly-owned subsidiary of Bloomberg L.P. ("BLP"). BLP provides BFLP with all global marketing and operational support and service for the Services and distributes the Services either directly or through a non-BFLP subsidiary in the BLP Countries. The Services include electronic trading and order-routing services, which are available only to sophisticated institutional investors and only where the necessary legal clearances have been obtained. BFLP, BLP and their affiliates do not provide investment advice or guarantee the accuracy of prices or information in the Services. Nothing on the Services shall constitute an offering of financial instruments by BFLP, BLP or their affiliates. BLOOMBERG, BLOOMBERG PROFESSIONAL, BLOOMBERG MARKETS, BLOOMBERG NEWS, BLOOMBERG ANYWHERE, BLOOMBERG TRADEBOOK, BLOOMBERG BONDTRADER, BLOOMBERG TELEVISION, BLOOMBERG RADIO, BLOOMBERG PRESS and BLOOMBERG.COM are trademarks and service marks of BFLP, a Delaware limited partnership, or its subsidiaries.

Table of Contents

Document Version History	5
INTRODUCTION.....	6
Introduction	7
EMSX API Programmable Support	8
EMSX API – High Level Concept	10
Installing the Desktop API SDK	10
Accessing EMSX API from Server	11
EMSX API – Server Side Request/Response	13
readme.txt.....	13
EMSX API – Application Structure	17
EMSX API – Data Paradigm	18
Bloomberg API Best Practices	19
EMSX API – Accessing the Test Environment	20
Programming References.....	21
PROGRAMMABLE.....	23
EMSX API – Programmable	24
EMSX API & CorrelationID	25
Connecting to BBCOMM and Creating a Session Object	25
Request/Response Services	26
Description of Request/Response Services.....	27
CFD & ODD LOT Flags	27
Custom Notes and Free Text Fields.....	28
Create an Order Request	29
Create an Order and Route Request	30
Create an Order and Route with Strategy Request.....	30
Create an Order and Route Manually Request	33
Modifying an Order Request.....	34
Modifying a Route Request.....	34
Modifying a Route with Strategy Request	35

Bloomberg

Creating a Route Request	36
Creating a Route with Strategy Fields Request	36
Creating a Route Manually Request	37
Cancelling a Route Request	38
Deleting an Order Request	38
Creating Basket from EMSX API Request	40
Trading on behalf of team members from TEAM_VIEW Request	40
Multi-Leg Options using EMSX API	41
Get All Field Meta Data Request	42
Get Field Meta Data Request	42
Get Teams Request	43
Get Brokers Request	43
Get Broker Strategies Request	44
Get Broker Strategy Information Request	44
Order Information Request	45
Route Information Request	45
Subscribing to Orders and Routes	46
Asynchronous Subscription	46
Description of Subscription Message	48
Description of Event Status Message	48
Description of Order Status	48
Description of Child Route Status	49
AIM Specific Fields	49
Using Subscription and Request/Response Messages	50
Python & EMSX API	71
How to setup the Python BLPAPI	71
MATLAB & EMSX API	78
Fills over EMSX API (/blp/emsx.history)	79
Failover	80
Time Zone in EMSX API	81
Extended Requests	81
FAQ	81
Accessing Field Meta Data	86

EXCEL	87
EMSX API – Excel Add-In	88
Getting Started with Excel Add-In	88
EMSX Blotter Actions – Excel Add-In	89
EMSX Staging, Order and Route Blotters – Excel Add-In	89
EMSX Settings – Excel Add-In	90
Broker Strategies / Algorithms – Excel Add-In	91
Short Selling – Excel Add-In	92
EMSX API Excel Add-In Fields	92
EMSX API Excel Add-In Use Cases	93
Troubleshooting EMSX API Excel Add-In	98
EMSX API – COM (Component Object Model)	100
Schema for Request / Response Messages	102

Document Version History

Date	Version	Author	Summary of Change
09/01/2010	v0.1	Jonathan Marks	Initial version
05/18/2011	v0.3	Raghupathy Srinivasan	Added / Revised C# Samples
05/19/2011	v0.4	Terrence C. Kim	Reorganized and redrafted the Programmer's guide
	v0.4	Piroon Tangnavarad	Updated XML Schema
06/01/2011	v0.5	Terrence C. Kim	Added page numbers incorporated user comments
06/02/2011	v0.6	Raghupathy Srinivasan	Incorporated R&D review comments
08/01/2011	v1.0	Terrence C. Kim	Version 1.0, out of draft state
08/08/2011	v1.1	Terrence C. Kim	Minor modifications
09/01/2011	v1.2	Terrence C. Kim	Clarified PG15,18
11/16/2011	v1.3	Terrence C. Kim	Fixed Handling Instruction Code in C#
04/12/2012	v2.0	Terrence C. Kim Richard Clegg	Added Broker Strategies, auto-activate and Excel Add-in information to this documentation.
03/04/2013	v2.0.5	Terrence C. Kim	Draft version before the release of v3.0. This includes MATLAB information and few enhanced code samples.
04/19/2013	v2.0.6	Terrence C. Kim	Bbapidemo.exe has change to BlpAPIDemo.exe.
04/01/2013	v2.1	Terrence C. Kim Piroon Tangnavarad Huan Li	Added details to creating baskets directly from EMSX API and trading on behalf of team members in TEAM_VIEW
09/01/2013	v3.0.4	Terrence C. Kim Richard Clegg Dave Butala	Updated Java Sample codes in WAPI<GO> and changed the entire code sample in the document to Java from C#. Included best practices, enriched the overall samples for using sub + req, and added MATLAB and Python information.
10/01/2013	v3.1	Terrence C. Kim Richard Clegg	Updated to new schema xxxEX and deprecated Generalxxxx on the request names. Added -99999 information for resetting limit price when using Modify requests
04/27/2015	v4.0	Terrence C. Kim Richard Clegg	Added details to access the existing EMSX API service from the server environment, updated new global support channel and deprecated dssupp@bloomberg.net , minimized code samples
8/5/2015	v4.1	Terrence C. Kim	Added details on how to access emapisvc & emapisvc_beta from server
8/17/2015	v4.1.1	Terrence C. Kim	Added odd lot details and CFD flag details
9/15/2015	v4.2	Terrence C. Kim	Added description of //blp/ emsx.history service call for fills information and new support page description along with multi-leg options trading via EMSX API.
11/16/2015	v4.3	Terrence C. Kim	Added description of EMSX_CUSTOM_NOTE element along with the details to free text character limits.

INTRODUCTION

Bloomberg

Introduction

This document is for developers who will use the Bloomberg EMSX API to develop custom applications. It is highly recommended that all users read from page 1-17 before moving onto the Excel or Programmable section. For any programmable API support questions, please log into <https://service.bloomberg.com>.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein. The sample codes are to demonstrate a particular function using EMSX API and should not be used or replicated in the production environment.

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 custom application in C++, C# (.NET), Python and Java. 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.

EMSX API users will need the following steps completed before using the API.

- 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 production environment.*
- For server side access, Serverapi.exe need to be installed, registered and permissioned
- 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.
- Download Bloomberg Desktop API v3 SDK from **WAPI <GO>**
- For Server Side API access, need signed EMSxNET Order Originator Agreement, install and register serverapi installer.

Customer Support Information

24 Hour Customer Support: Call Bloomberg 24 hour support and ask for EMSI support		
Americas	United States	+1 212 318 2000
	Brazil	+55 11 3048 4500
Europe	United Kingdom	+44 20 7330 7500
	France	+33 1 5365 5000
	Switzerland	+41 22 317 9200
Asia Pacific	Hong Kong	+852 2977 6000
Press HELP key twice while on EMSX screen OR use https://service.bloomberg.com		

EMSX API Programmable Support

For a new user, you will need to first start by creating the account in <https://service.bloomberg.com> and select "Request a new account".

Bloomberg the Company & its Products | Bloomberg Anywhere Remote Login | Bloomberg Terminal Request a Demo

Bloomberg Welcome to the Bloomberg Customer Service Center
Manage your contracts, exchanges, billing, and enterprise products. Order new services, access documentation, create support tickets and receive relevant notifications all in one streamlined site.

Sign in to your account.

Username
[Input Field]

Password
[Input Field]

Language
English

☐ Remember my username

SIGN IN

[Forgot Password?](#)
[Request a new account.](#)

Need help? [Call customer support.](#)

Bloomberg

Fill out the details on Account Registration and select B-Pipe, select the role as *Technical Contact*, and insert Customer #. The Customer # can be found in the terminal by typing **IAM<GO>**.

Bloomberg CUSTOMER SERVICE CENTER

Account Registration

User Details

Username * JohnDoe

First Name * John

Last Name * Doe

Email * j.doe@email.com

Phone # * 123-456-7890

Fax #

Bloomberg Login JDOE99

Product Information

Please select the product you would like to

☐ Data License or BVAL - Derivatives

☒ B-Pipe

Select a role: * Technical Contact

☐ BPID ☒ Customer #

☐ Server API

☐ BVAL - Fixed Income

* Required field

Register Back

Click Register on the bottom left to finish the registration.

Bloomberg

EMSX API – High Level Concept

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 schemas 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.

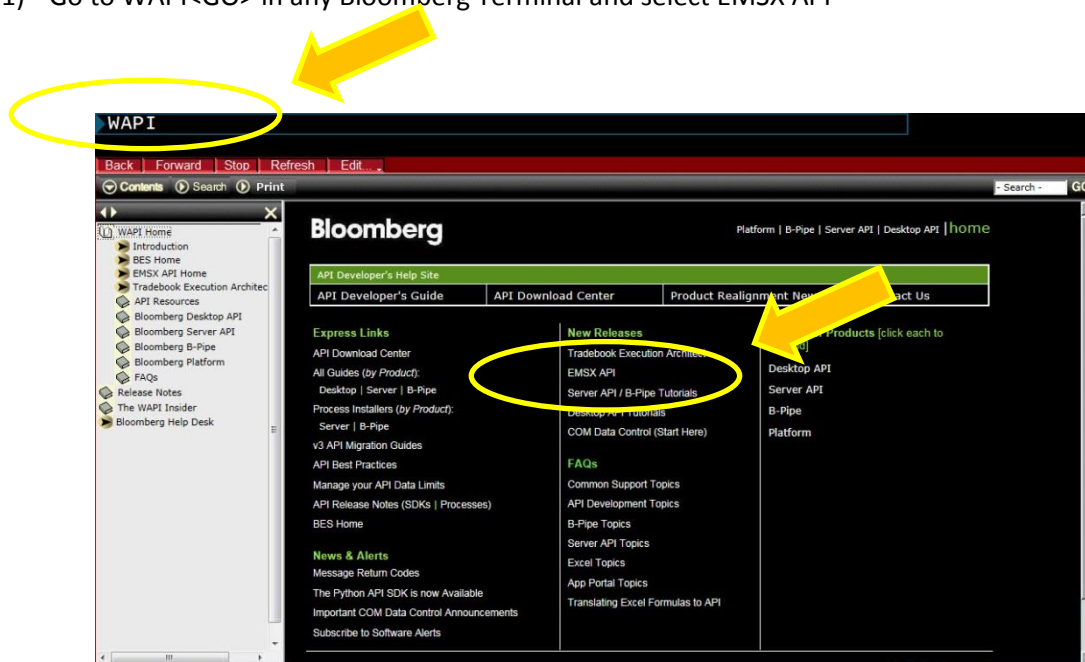
Installing the Desktop API SDK

For details of the Desktop API SDK, please refer to the API Download Center in **WAPI<GO>** in your Bloomberg terminal.

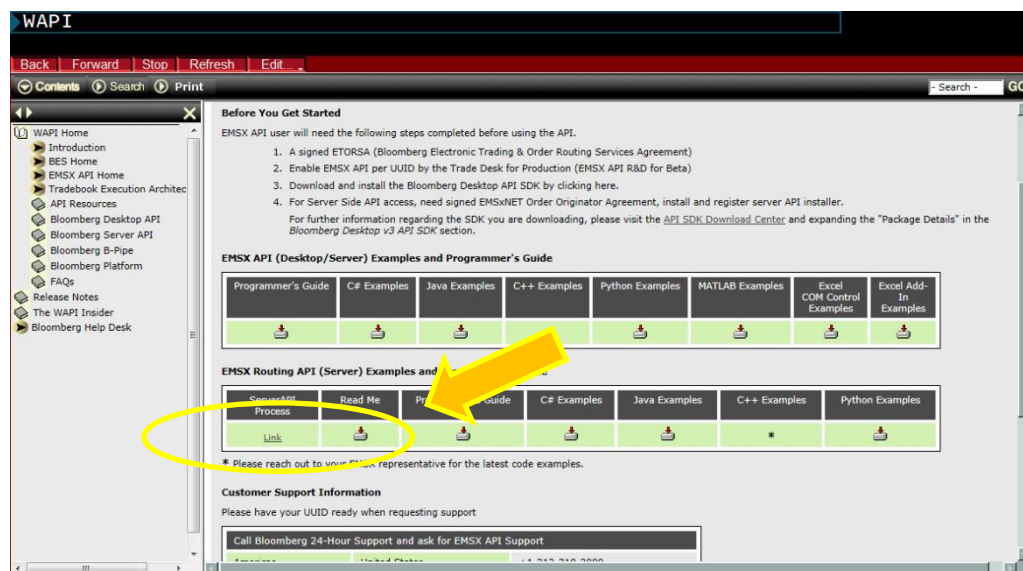
Bloomberg

Accessing EMSX API from Server

- 1) Go to WAPI<GO> in any Bloomberg Terminal and select EMSX API

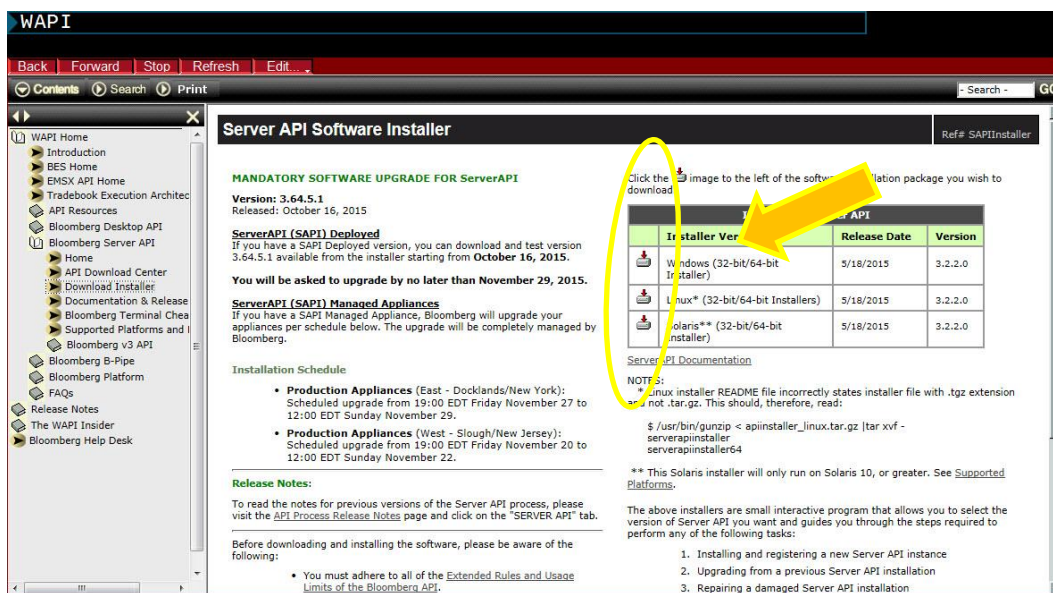


- 2) Go to EMSX Routing API (Server) → Server API Process section and click on the Link.

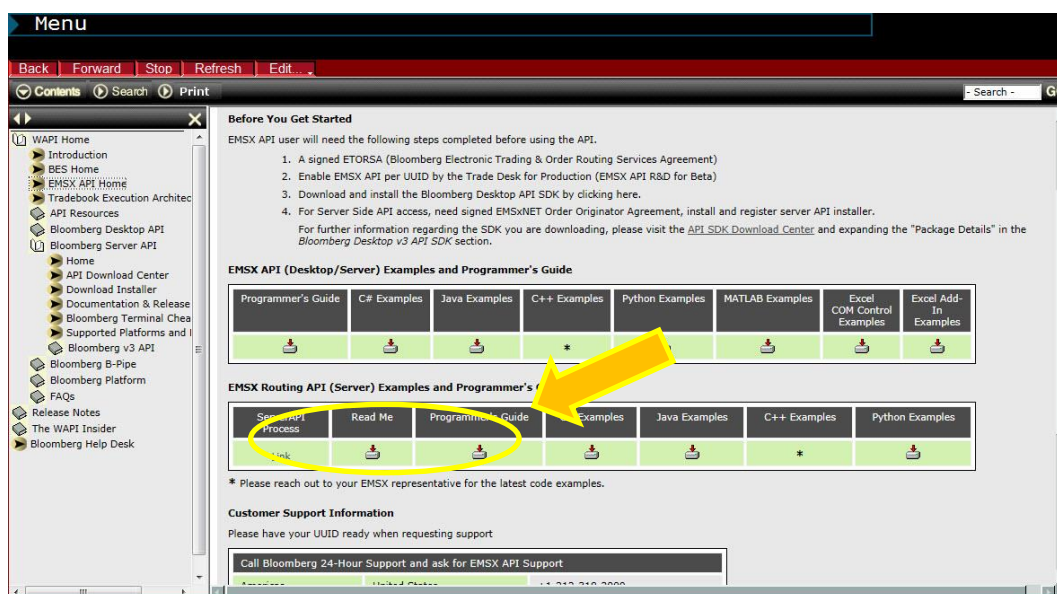


Bloomberg

- 3) Select between Windows/Linux/Solaris installer and install to the server.



- 4) Refer to the [following](#) link or refer to Read Me under EMSX Routing API (Server) section.



Once installation is completed, reach out to your local EMSX Implementation team in your region for the next step.

EMSX API – Server Side Request/Response

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

CancelRoute	RouteEx
CreateOrder	RouteManuallyEx
CreateOrderAndRouteEx	ModifyRouteEx
DeleteOrder	GetBrokers
GroupRouteEx	GetBrokerStrategies
ModifyOrder	GetBrokerStrategyInfo

Any other requests will return the following error:

"[Obsolete request type](#): " << request_type

readme.txt

```
* Please follow the following steps to install and register the installer with  
Enterprise Solutions.  
*/
```

Bloomberg Enterprise Solutions Installer Examples
1) Use the following command for Private IP connection

```
./serverapiinstaller64 -i 208.134.161.176
```

1a) Use the following command for Internet connection

```
./serverapiinstaller64 -i apibeta1.bloomberg.net
```

2) Installer will show the following message:

```
logging to /home12/InstallPath/install.2014120512.092756.log  
  
Bloomberg Server API Installer for Linux (64-bit) Version 3.0.4.0  
  
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.
```

Bloomberg

3) Installer will show the following message:

```
Would you like to continue? (Y/N): y
Checking connectivity to Bloomberg ...
Connecting to [ Hostname = apibeta1.bloomberg.net Port = 8194 ]: succeeded.
Installation path:          '/opt/local'
```

4) Installer will show the following message:

```
Use this path? (Y/N/Q): n
Enter installation path: /home/YourInstallPath
Installation path:          '/home/YourInstallPath'
```

5) Installer will show the following message:

```
Use this path? (Y/N/Q): y
Newer Version of loader available. Upgrading loader to version 3.2.1.0.
Could not parse configuration file serverapiinstaller64.cfg, will prompt for
install info logging to /tmp/install.2014120512.092823.log
Bloomberg ECD Installer for Linux (64-bit) Version 3.2.1.0
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.
```

Bloomberg

6) Installer will show the following message:

```
Would you like to continue? (Y/N): y
Checking connectivity to Bloomberg ...
Connecting to [ Hostname = apibeta1.bloomberg.net Port = 8194 ]: succeeded.
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
```

7) Installer will show the following message:

```
Please enter selection: 1
Downloading latest installer ... done.
logging to /tmp/install.2014120512.093429.log
Beginning new install ...
Versions available for blpddm

1) 3.48.9.1      Linux64      blpddm - unmanaged platform

0) Quit
```

8) Installer will show the following message:

```
Please enter version of blpddm that you want to install: 1
Downloading blpddm components ...
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.): Your Company

Department Name (e.g., Equity Trading): FX
```

Bloomberg

```
Creating certificate ... done.  
Registering server ... done.  
done.
```

9) 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: 839f-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx  
This key was also saved in regkey.txt in the blpddm root directory.  
blpddm installation completed. Press ENTER to quit:
```

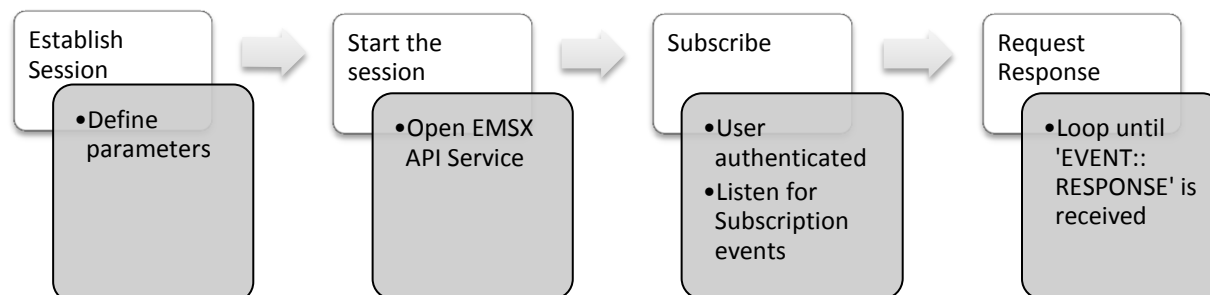
10) Contact your Bloomberg Implementation Representative to register the key.

```
#Comment: The installation will be in '/home/YourInstallPath/blpddm'.
```

```
/* Once the registration process is completed. EMSX Implementation team will  
* assist with configuring the Server Side EMSX API with various execution  
* destinations per client request.  
*/
```


EMSX API – Application Structure

Today, the EMSX API is available on the same desktop where a Bloomberg Anywhere license is installed and as an independent server side offering. The Desktop API 3.0 is used when the end-user application resides on the same machine as the installed BLOOMBERG PROFESSIONAL service and connects to the local Bloomberg Communications Server (BBCOMM) to obtain data from the Bloomberg EMSX service.

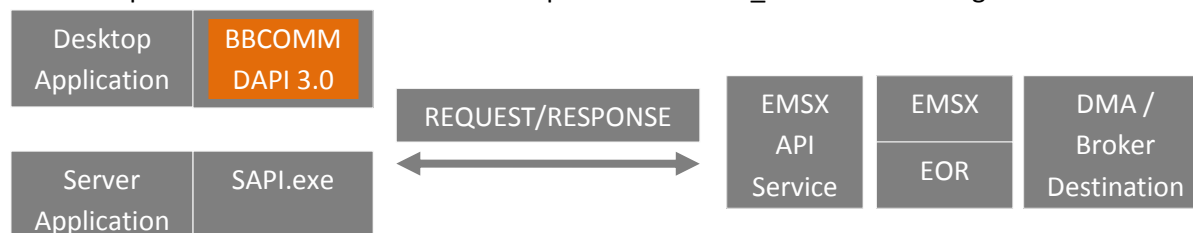


There are two types of services, Subscription and Request/Response services.

Subscription: The client's desktop application registers with Bloomberg for specific events and Bloomberg pushes data to the client application when events are triggered. A subscription establishes a channel through which you will receive events. An event here is defined as any messages published by the server. Events continue to be delivered to the client application until the Subscription is explicitly cancelled by the application, or as the application exits. In order to subscribe successfully to the service, the user must be logged into their Bloomberg session.

- The following will cause an API session (BBCOMM) to terminate.
 - ✓ After 24 hours of non-login into a Bloomberg terminal
 - ✓ The Bloomberg user logs into another terminal
 - ✓ A different Bloomberg user logs into the current terminal
 - ✓ Bloomberg user logs into their mobile application
 - ✓ In addition, software updates will cause the BBCOMM to be terminated. The terminal launches these when it detects that it is idle.

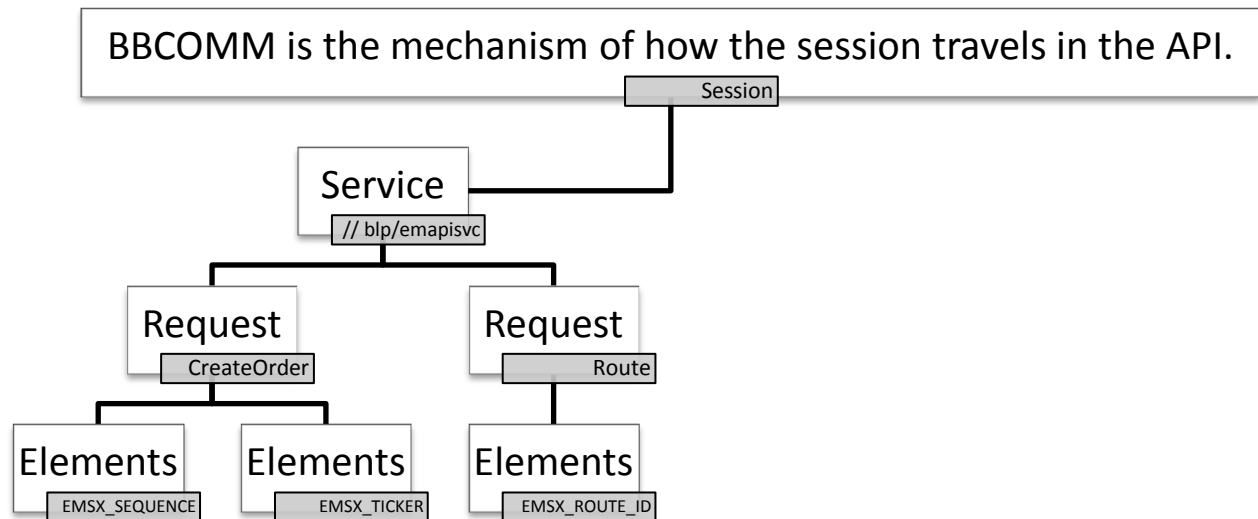
Request/Response: The client application requests data and Bloomberg returns data. In this case, the user application requests the data or performs certain actions by issuing a Request. Once this request is processed, exactly one Event of type RESPONSE is returned. This RESPONSE indicates that the Request has been completed. The EMSX service does not produce PARTIAL_RESPONSE messages.



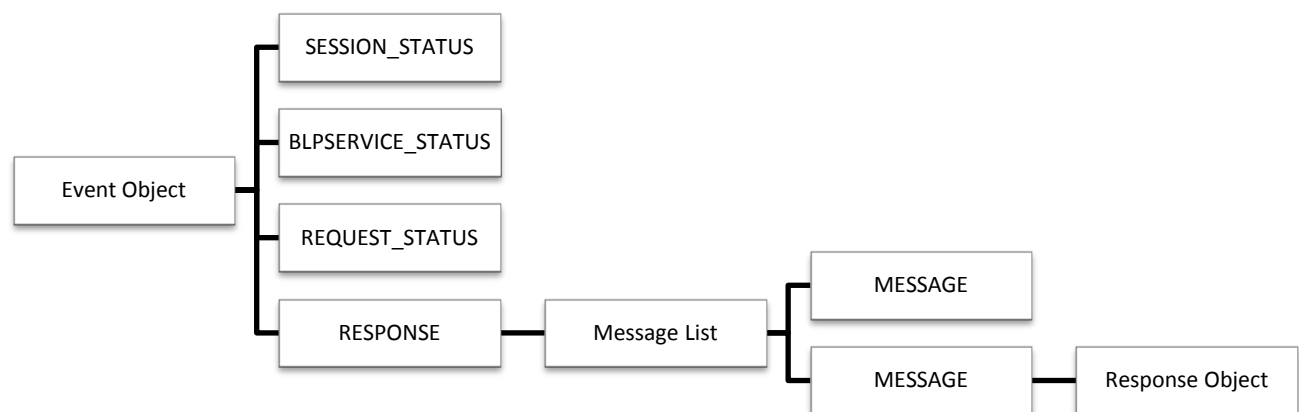
EMSX API – Data Paradigm

A Subscription is created and this results in a stream of updates being delivered as SUBSCRIPTION_DATA events. This is used to request real time information only. In EMSX API, there are certain fields that are labeled as static. Static fields are only updated in INIT_PAINT or NEW_ORD_UPD messages.

Request/Response data is serviced by issuing a request and returned in a RESPONSE event. Unlike the standard Desktop API 3.0, EMSX API will not return PARTIAL_RESPONSE.



EMSX API Events



Bloomberg

Bloomberg API Best Practices

Ticker

When submitting the ticker information, it is always best to use the full Bloomberg ticker 'Ticker + Exchange + Asset Class' to enhance the performance and avoid any issues. (e.g. 'AAPL US Equity' or 'CLH4 NYM Comdty')

Hash Table/Associative Array

Due to the event driven nature of the Bloomberg API, it is highly recommended for developers to implement a data structure that can map keys to values. Following lists the terms often used for this in different programming languages. Associative array, hash table, hash maps, dictionary, dictionary object, list, mapping, or key-value list.

Subscription

Although the Bloomberg API supports both synchronous and asynchronous communication modes, asynchronous is most of the time the better-suited communication mode for the design of the application using the Bloomberg API. This is simply because the asynchronous method ensures the Bloomberg backend to process the results as fast as it is possible to do.

The user will always need to manage how much and how fast the user is sending to the backend. If the user is receiving timeouts, this is an indication that too much is being sent too fast to the backend. There is no magical work around for this limitation.

Request/Response

In EMSX API, EMSX_REQUEST_SEQ can be added to every request coming from the user. The purpose of this unique user assigned sequence number to each API request is for Bloomberg to add the details to the database for comparison with the existing records. If the sequence number were already sent over to the backend, then the backend processor of EMSX would drop this request and return an error response to the user. This is to prevent duplicate requests being sent during a system outage of any type.

EMSX_REQUEST_SEQ should be a unique ID, which consists of 64-bit integer. This ID can be reset every day of the users date but we recommend the user reset this once a week when possible. The number generated should be unique per serial number of the Bloomberg terminal.

For best practice, Bloomberg API users should gather the requests together up to a point before sending the requests to the backend of EMSX. Chunking is the best way since this allows for the potential of different back end machines handling the individual chunks.

Bloomberg

Sessions

For sessions, single session is always preferable over having multiple sessions. Although the Bloomberg API is completely thread safe and multiple sessions are possible, this is normally for handling different kinds of subscriptions, requests and responses and not for handling large amounts of requests.

The user will still only have a finite number of clock cycles on the machine and multiple threads and multiple sessions does not reduce the amount of work that needs to be done. This approach can possibly introduce an unforeseen overhead as well.

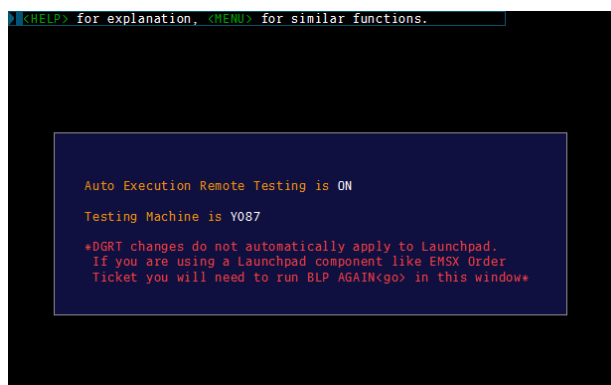
EMSX API – Accessing the Test Environment

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 BBCOMM 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**.

In the Excel Add-In, a user can switch to different environment by toggling the environment button on the bottom of the Excel Add-In Control Panel.

During the initial setup, the EMSX Trade Desk in each region will enable the test environment for each UUID. In order to access EMSX from the beta environment for testing, the user will need to remotely log into test environment by typing **DGRT Y087 <GO>** inside the terminal.

Inside the Bloomberg Terminal type **DGRT Y087<GO>** {Y (number zero) 87}



```

[<HELP> for explanation, <MENU> for similar functions.]

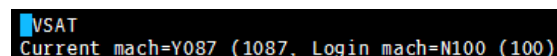
Auto Execution Remote Testing is ON
Testing Machine is Y087

*DGRT changes do not automatically apply to Launchpad.
If you are using a Launchpad component like EMSX Order
Ticket you will need to run BLP AGAIN<go> in this window*
```

This command allows the particular terminal window to log into the beta environment.

Please note, when a user is remote into the beta environment it only affects that particular terminal screen and the other screens will not be affected by the DGRT command.

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



```

VSAT
Current mach=Y087 (1087, Login mach=N100 (100)
```

Bloomberg

To get back to production type **DGRT 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 lot slower than the production environment.


To launch the ticket using limited functional terminal will require different command once inside the Y087 environment.

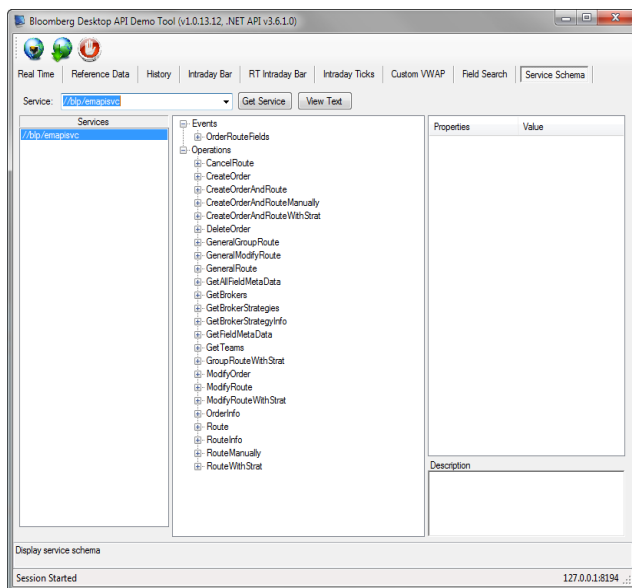
EMSX /NOLP <GO> - This command disables Launchpad ticket from EMSX.

Programming References

Goto WAPI<GO> and select API Download Center and click the relevant download file and drag the APIv3 folder into you C:\blp\API directory.

Once the Bloomberg API SDK has been downloaded and installed, the user should get familiar with the BlpAPIDemo.exe. This can usually be found in C:\blp\API\APIv3\bin. Once in this directory the user can launch the application by double clicking the **BlpAPIDemo.exe** application.

Once the application is launched, the user can establish the Server Connection by clicking on the  button. Once the user is inside the application, click on the Service Schema tab, which is the last tab on the far right. Once the user is inside the Service Schema, user can access the live schema of EMSX API by typing service type **//blp/emapisvc_beta** for development environment and type **//blp/emapisvc** for production.



This will expose all the fields that a user can call from each environment for Events and Operations.

Events outline the subscriptions field available. *Operations* outline the *Request and Response* field types.

Bloomberg

** Discrepancies in Field Names, following are same fields under slightly different field name.

Subscription	Request / Response
EMSX_ORD_REF_ID	EMSX_ORDER_REF_ID
EMSX_TRAD_UUID	EMSX_TRADER_UUID

PROGRAMMABLE

Bloomberg

EMSX API – Programmable

The programmable API provides developers with access to EMSX data via a number of programming languages. It can be used independently of the EMSX Excel add-in, or as a complement. The API provides 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 further 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 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 guarantee 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 the user has an option of using EMSX_ORD_REF_ID in subscription by supplying the EMSX_ORDER_REF_ID in your 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.

Bloomberg

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.

EMSX API & CorrelationID

CorrelationID ties the subscriptions and request response messages. Subscriptions have CorrelationID. 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 CorrelationIDs are set when you send the request or submit the subscription. The CorrelationIDs 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. For example, the user can have '1' as the CorrelationID for *mktdata* subscription, '2' for orders and '3' for routes. For the *CreateOrder* call, the user can use the specified ID for the CorrelationID, and then the user can directly associate the EMSX_SEQUENCE the user gets back from the response to the ID contained in the *.correlationID* property of the message.

Connecting to BBCOMM and Creating a Session Object

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.

Request/Response Services

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:

- i. Create a session (if session does not yet exist).
- ii. Connect session to `//blp/emapisvc_beta` or `//blp/emapisvc` service and start it.
- iii. Fetch a service object from the session representing `emapisvc`.
- iv. Use the service object from above to create a Request object of the desired type
- v. Send request object via `sendRequest` method of session object, pass object of type `EventQueue` to the `sendRequest`.
- vi. 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. Following table outlines the Request and Response Message Types.

Action	Request Type	Underlying Request	Response Message Class
Create Order	CreateOrder	CreateOrderRequest	OrderStaticData
Create Order and Route (In 1 step)	CreateOrderAndRoute	CreateOrderRequest	RouteStaticData
Route Order	Route	RouteOrderRequest	RouteStaticData
Modify Order	ModifyOrder	ModifyOrderRequest	OrderStaticData
Delete Order	DeleteOrder	DeleteOrderRequest	DeleteOrderResponse
Modify Route	ModifyRoute	ModifyRouteRequest	RouteStaticData
Cancel Route	CancelRoute	CancelRouteRequest	CancelRouteResponse
Get Teams	GetTeams	GetTeamsRequest	GetTeamsResponse
Get Order info	OrderInfo	OrderInfoRequest	OrderInfoResponse
Get Route info	RouteInfo	RouteInfoRequest	RouteInfoResponse
Get Brokers	GetBrokers	GetBrokersRequest	GetBrokersResponse
Get Broker Strategies	GetBrokerStrategies	GetBrokerStrategiesRequest	GetBrokerStrategiesResponse
Get Broker Strategy Information	GetBrokerStrategyInfo	GetBrokerStrategyInfoRequest	GetBrokerStrategyInfoResponse
Create Order and Route with Strategy	CreateOrderAndRouteWithStrat	CreateOrderWithStrategyRequest	RouteStaticData
Route Order with Strategy	RouteWithStrat	RouteOrderWithStrategyRequest	RouteStaticData
Modify Route with Strategy	ModifyRouteWithStrat	ModifyRouteWithStrategyRequest	RouteStaticData

Description of Request/Response Services

EMSX API supports the following Request/Response services:-

Request Name	Action
CreateOrder	Create an order. Response contains new sequence number of failure message.
CreateOrderAndRoute	Create a new Order and route it in a single step
CreateOrderAndRouteManually	Create the order, and notify EMSX that it is routed. This is generally used for phone orders, where the routing is external to EMSX.
CreateOrderAndRouteWithStrat	Creates a new Order and Route with strategies in a single step
DeleteOrder	Deletes an existing Order in EMSX
Route	Route Order in EMSX
RouteWithStrat	Route Order with strategy fields
CancelRoute	Cancel Route in EMSX
ModifyOrder	Modifies an existing Order in EMSX
ModifyRoute	Modifies an existing Route in EMSX
ModifyRouteWithStrat	Modifies an existing Route with broker strategies in EMSX

- The response for any action contains only minimal information about the order or route. In case of success, the return message contains the EMSX generated order number and EMSX generated route number. In case of an error, it contains an error code and an error message.
- All the details of the order and or route are sent back to the application via the subscription mechanism. If the action succeeds and the new order/route is created or modified all the details on that order are published to any earlier created subscription.
- Note: This synchronous approach may prove to be a bottleneck and it may be a better choice to follow the asynchronous model as described in the asynchronous subscription approach.

CFD & ODD LOT Flags

To indicate CFD orders or to flag an Odd Lot in EMSX API works as following:

Elements	Action
EMSX_CFD_FLAG	This element is to flag a particular order as CFD. (Contract for Difference) 0 = not flagged 1 = flagged
EMSX_ODD_LOT	An odd lot is a quantity of stock that amounts to less than 100 shares. A deal involving 100 shares or more is considered a round-lot transaction. 0 = not odd lot (it won't fill odd lots) 1 = odd lot

Custom Notes and 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 internal only workflow where the others can be used to communicate with the various execution counterparts.

Request Element Name	Internal Only	Description
EMSX_ACCOUNT	No	30-character free text field (29 + 1 check digit) and available on order and route subscription services, this is usually mapped to tag 1 on the outbound FIX message.
EMSX_NOTES	No	44-character free text field (43 + 1 check digit) and available on order and route subscription services, this is usually mapped to tag 58 on the outbound FIX message.
EMSX_ORDER_REF_ID	No	16-character field (15 + 1 check digit) used for EMSX_ORD_REF_ID in order subscription service.
EMSX_TRADER_NOTES	Yes	44-character field (43+1 check digit) and is only available as view only on order subscription service and cannot be edited or inserted using EMSX API. <u>This is for internal use only and will not be communicated with external execution counterparties.</u>
EMSX_CUSTOM_NOTE1	No*	80-character free text field (79 + 1 check digit) and is NOT available via subscription services.
EMSX_CUSTOM_NOTE2	No*	80-character free text field (79 + 1 check digit) and is NOT available via subscription services.
EMSX_CUSTOM_NOTE3	No*	80-character free text field (79 + 1 check digit) and is NOT available via subscription services.
EMSX_CUSTOM_NOTE4	No*	80-character free text field (79 + 1 check digit) and is NOT available via subscription services.
EMSX_CUSTOM_NOTE5	No*	80-character free text field (79 + 1 check digit) and is NOT available via subscription services.

**If the custom tags are not set via FIX, this will stay internal to the buy-sides and will not be sent to the execution counterparts. Please inquire your EMSX account managers for details on a particular broker codes setup to check if a particular broker code has custom FIX tags setup or not.*

Create an 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.

```
# EMSXCreateOrderRequest.py

import blpapi
import argparse

ERROR_INFO      = blpapi.Name("ErrorInfo")
CREATE_ORDER    = blpapi.Name("CreateOrder")

d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
...
    request = service.createRequest("CreateOrder")

    request.set("EMSX_TICKER", options.EMSX_TICKER)
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
    request.set("EMSX_TIF", options.EMSX_TIF)
    request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTIONS)
    request.set("EMSX_SIDE", options.EMSX_SIDE)

    if options.EMSX_ACCOUNT:      request.set("EMSX_ACCOUNT", options.EMSX_ACCOUNT)
    if options.EMSX_LIMIT_PRICE:  request.set("EMSX_LIMIT_PRICE", options.EMSX_LIMIT_PRICE)
    if options.EMSX_BROKER:       request.set("EMSX_BROKER", options.EMSX_BROKER)
    if options.EMSX_CFD_FLAG:     request.set("EMSX_CFD_FLAG", options.EMSX_CFD_FLAG)

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

    requestID = blpapi.CorrelationId(1)

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

The mandatory fields for the *CreateOrder* requests are the following:-

Mandatory Fields to <i>CreateOrder</i> Request			
EMSX_TICKER	EMSX_SIDE (Enumeration Field)	EMSX_AMOUNT	EMSX_ORDER_TYPE (Enumeration Field)
EMSX_BROKER	EMSX_TIF (Enumeration Field)	EMSX_HAND_INSTRUCTION	

They can only accept values specified in the corresponding enumerated types and depending on the values of the fields, other requests fields may become mandatory. The following are some of the examples:-

Bloomberg

- i. EMSX_LIMIT_PRICE becomes mandatory if EMSX_ORDER_TYPE takes values: LMT, SL, and LOC. Failure to specify value for this field will result in returned error message.
- ii. For all other values for EMSX_ORDER_TYPE field EMSX_LIMIT_PRICE is ignored.
- iii. EMSX_GTD_DATE (integer in format *yyyymmdd*) becomes mandatory if field EMSX_TIF is assigned value GTD.
- iv. EMSX_STOP_PRICE is mandatory when EMSX_ORDER_TYPE is SL or ST.

Create an Order and Route Request

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

```
# EMSXCreateOrderAndRouteRequest.py
...
    request = service.createRequest("CreateOrderAndRouteEx")

    request.set("EMSX_TICKER", options.EMSX_TICKER);
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT);
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE);
    request.set("EMSX_BROKER", options.EMSX_BROKER);
    request.set("EMSX_TIF", options.EMSX_TIF);
    request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTION);
    request.set("EMSX_SIDE", options.EMSX_SIDE);

    if options.EMSX_ACCOUNT:      request.set("EMSX_ACCOUNT",options.EMSX_ACCOUNT);
    if options.EMSX_LIMIT_PRICE:  request.set("EMSX_LIMIT_PRICE", options.EMSX_LIMIT_PRICE);
    if options.EMSX_CFD_FLAG:     request.set("EMSX_CFD_FLAG", options.EMSX_CFD_FLAG);
    if options.EMSX_SETTLE_DATE:  request.set("EMSX_SETTLE_DATE", options.EMSX_SETTLE_DATE);
    if options.EMSX_STOP_PRICE:   request.set("EMSX_STOP_PRICE", options.EMSX_STOP_PRICE);

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID)

    timeout = 10000
...
```

Create an Order and Route with Strategy Request

Creating an order and routing with strategy requires the user to create a request from the service object of type *CreateOrderAndRouteWithStrat* and fill in the required fields before submitting the request.

Mandatory fields for the *CreateOrderAndRoute* requests are the following.

Mandatory Fields to <i>CreateOrderAndRoute</i> Request			
EMSX_SIDE (Enumeration Field)	EMSX_AMOUNT	EMSX_TIF (Enumeration Field)	EMSX_ORDER_TYPE (Enumeration Field)
EMSX_BROKER	EMSX_TICKER	EMSX_HAND_INSTRUCTION	

Bloomberg

The user will first need to request *GetBrokers* to get all the brokers the user is enabled for, returned in response. Subsequently the user can then request *GetBrokerStrategies* to get all the broker strategies user is enabled for that particular broker code. Lastly, *GetBrokerStrategyInfo* will get all the fields for the provided broker strategy in the particular order in which they need to be submitted in *CreateOrderAndRouteWithStrat*, *CreateOrderAndRouteEx* and *RouteWithStrat* requests.

```
/* Pseudo Code */

/*Get strategy Details using GetBrokers, GetBrokerStrategies and GetBrokerStrategyInfo*/

Request request = m_service.CreateRequest ("GetBrokers");
    request.set("EMSX_TICKER", "AAPL US Equity"); // Ticker to identify asset class

SubmitRequest (request);

// Response from GetBrokers with list of broker codes enabled for the UUID
GetBrokers = {
    EMSX_BROKERS[] = {
        BB
        API
        BMTB
    }
}

// GetBrokerStrategies
Request request = m_service.CreateRequest ("GetBrokerStrategies");
    request.set("EMSX_BROKER", "API");           // Broker code
    request.set("EMSX_TICKER", "AAPL US Equity"); // Ticker to identify asset class

SubmitRequest (request);

// Response from GetBrokerStrategies for the given broker code and asset class
// enabled for the UUID
GetBrokerStrategies = {
    EMSX_STRATEGIES[] = {
        DMA
        IS
        POV
        TWAP
        VWAP
    }
}

// GetBrokerStrategyInfo
Request request = m_service.CreateRequest ("GetBrokers");
    request.set("EMSX_BROKER", "API");           // Broker code
    request.set("EMSX_STRATEGY", "VWAP");        // Strategy name
    request.set("EMSX_TICKER", "AAPL US Equity"); // Ticker to identify asset class

SubmitRequest (request);

// Response from GetBrokerStrategyInfo for the given strategy name will send back an array of
// the following.
GetBrokerStrategyInfo = {
    EMSX_STRATEGY_INFO[] =
```

Bloomberg

```
    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 =
    }
}
```

```
# EMSXCreateOrderAndRouteWithStratRequest.py
...
request = service.createRequest("CreateOrderAndRouteWithStrat")

request.set("EMSX_TICKER", options.EMSX_TICKER);
request.set("EMSX_AMOUNT", options.EMSX_AMOUNT);
request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE);
request.set("EMSX_BROKER", options.EMSX_BROKER);
request.set("EMSX_TIF", options.EMSX_TIF);
request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTION);
request.set("EMSX_SIDE", options.EMSX_SIDE);

if options.EMSX_SETTLE_DATE:    request.set("EMSX_SETTLE_DATE", options.EMSX_SETTLE_DATE);
if options.EMSX_STOP_PRICE:    request.set("EMSX_STOP_PRICE", options.EMSX_STOP_PRICE);

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

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

params = options.EMSX_STRATEGY_PARAMS.split(",")
for param in params:
    parts= param.split("=",1)
    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR",1 if parts[1]=="" else 0)
    data.appendElement().setElement("EMSX_FIELD_DATA",parts[1])

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

requestID = blpapi.CorrelationId(1)

session.sendRequest(request, correlationId=requestID)

timeout = 10000
...
```


Create an Order and Route Manually Request

Creating an order and routing manually requires the user to create a request from the service object of type *CreateOrderAndRouteManually* and fill in the required fields before submitting the request. This is generally used for phone orders, where the routing is external to EMSX API.

```
# EMSXCreateOrderAndRouteManuallyRequest.py
...
    request = service.createRequest("CreateOrderAndRouteManually")

    request.set("EMSX_TICKER", options.EMSX_TICKER);
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT);
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE);
    request.set("EMSX_BROKER", options.EMSX_BROKER);
    request.set("EMSX_TIF", options.EMSX_TIF);
    request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTION);
    request.set("EMSX_SIDE", options.EMSX_SIDE);

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID)

    timeout = 5000
...
```

Modifying an Order Request

Modifying an order requires the user to create a request from the service object of type *ModifyOrder* and fill in the required fields before submitting the request.

In cases where a user needs to modify an order from EMSX_ORDER_TYPE 'LMT' to 'MKT', the user needs to reset the limit price with -99999. This would clear out the limit price when modifying limit order to market order.

```
# EMSXModifyOrderRequest.py
...
    request = service.createRequest("ModifyOrder")

    request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
    request.set("EMSX_TICKER", options.EMSX_TICKER)
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
    request.set("EMSX_TIF", options.EMSX_TIF)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 500
...
```

Modifying a Route Request

Modifying a route requires the user to create a request from the service object of type *ModifyRoute* and fill in the required fields before submitting the request.

```
# EMSXModifyRouteRequest.py
...
    request = service.createRequest("ModifyRoute")

    request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
    request.set("EMSX_ROUTE_ID", options.EMSX_ROUTE_ID)
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
    request.set("EMSX_TICKER", options.EMSX_TICKER)
    request.set("EMSX_TIF", options.EMSX_TIF)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 500
...
```

Modifying a Route with Strategy Request

Modifying a route with strategy requires the user to create a request from the service object of type *ModifyRouteWithStrat* and fill in the required fields before submitting the request.

```
# EMSXModifyRouteWithStratRequest.py
...
    request = service.createRequest("ModifyRouteWithStrat")

    request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
    request.set("EMSX_ROUTE_ID", options.EMSX_ROUTE_ID)
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
    request.set("EMSX_TICKER", options.EMSX_TICKER)
    request.set("EMSX_TIF", options.EMSX_TIF)

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

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

    params = options.EMSX_STRATEGY_PARAMS.split(",")
    for param in params:
        parts= param.split("=",1)
        indicator.appendElement().setElement("EMSX_FIELD_INDICATOR",1 if parts[1]=="" else 0)
        data.appendElement().setElement("EMSX_FIELD_DATA",parts[1])

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 500
...
```

Creating a Route Request

Creating a route requires the user to create a request from the service object of type *Route* and fill in the required fields before submitting the request.

```
# EMSXRouteRequest.py
...
    request = service.createRequest("Route")

    request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
    request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
    request.set("EMSX_BROKER", options.EMSX_BROKER)
    request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTIONS)
    request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
    request.set("EMSX_TICKER", options.EMSX_TICKER)
    request.set("EMSX_TIF", options.EMSX_TIF)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 500
...
```

Creating a Route with Strategy Fields Request

Creating a route with strategy requires the user to create a request from the service object of type *RouteWithStrat* and fill in the required fields before submitting the request.

The following is an example of *RouteWithStrat* request. The user first request *GetBrokers* to get all the brokers the user is enabled for, returned in the response. Subsequently the user can then request *GetBrokerStrategies* to get all broker strategies user is enabled for, returned in the response. Lastly, *GetBrokerStrategyInfo* will get all the fields for the provided broker strategy in the particular order in which they need to be submitted in *CreateOrderAndRouteWithStrat* or *RouteWithStrat*.

```
# EMSXRouteWithStratRequest.py

import blpapi
import argparse

ERROR_INFO      = blpapi.Name("ErrorInfo")
ROUTE_WITH_STRAT = blpapi.Name("RouteWithStrat")

d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
...
    request = service.createRequest("RouteWithStrat")

    request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
```

Bloomberg

```
request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
request.set("EMSX_BROKER", options.EMSX_BROKER)
request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTION)
request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
request.set("EMSX_TICKER", options.EMSX_TICKER)
request.set("EMSX_TIF", options.EMSX_TIF)

...

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

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

params = options.EMSX_STRATEGY_PARAMS.split(",")
for param in params:
    parts= param.split("=",1)
    indicator.appendElement().setElement("EMSX_FIELD_INDICATOR",1 if parts[1]=="" else 0)
    data.appendElement().setElement("EMSX_FIELD_DATA",parts[1])

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

requestID = blpapi.CorrelationId(1)

session.sendRequest(request, correlationId=requestID)

timeout = 10000

...
```

Creating a Route Manually Request

Creating a route manually requires the user to create a request from the service object of type *RouteManually* and fill in the required fields before submitting the request.

```
# EMSXRouteManuallyRequest.py
...
request = service.createRequest("RouteManually")

request.set("EMSX_SEQUENCE", options.EMSX_SEQUENCE)
request.set("EMSX_AMOUNT", options.EMSX_AMOUNT)
request.set("EMSX_BROKER", options.EMSX_BROKER)
request.set("EMSX_HAND_INSTRUCTION", options.EMSX_HAND_INSTRUCTIONS)
request.set("EMSX_ORDER_TYPE", options.EMSX_ORDER_TYPE)
request.set("EMSX_TICKER", options.EMSX_TICKER)
request.set("EMSX_TIF", options.EMSX_TIF)

...
```

Cancelling a Route Request

Cancelling a route requires the user to create a request from the service object of type *CancelRoute* and fill in the required fields before submitting the request.

```
# EMSXCancelRouteRequest.py

import blpapi
import argparse

ERROR_INFO      = blpapi.Name("ErrorInfo")
CANCEL_ROUTE    = blpapi.Name("CancelRoute")

d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
...
    request = service.createRequest("CancelRoute")

    routes = request.getElement("ROUTES")

    route = routes.appendElement()

    route.getElement("EMSX_SEQUENCE").setValue(options.EMSX_SEQUENCE)
    route.getElement("EMSX_ROUTE_ID").setValue(options.EMSX_ROUTE_ID)

    if options.EMSX_TRADER_UUID:      request.set("EMSX_TRADER_UUID", options.EMSX_TRADER_UUID)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Deleting an Order Request

Deleting an order requires the user to create a request from the service object of type *DeleteOrder* and fill in the required fields before submitting the request.

The following is an example of *DeleteOrder* request. This request is different from the other requests since the *DeleteOrder* supports cancellation of multiple orders in single request as follows:-

```
# EMSXDeleteOrderRequest.py

import blpapi
import argparse

ERROR_INFO      = blpapi.Name("ErrorInfo")
DELETE_ORDER    = blpapi.Name("DeleteOrder")

d_service="//blp/emapisvc_beta"
```

Bloomberg

```
d_host="localhost"
d_port=8194
...
    request = service.createRequest("DeleteOrder")

    sequences = request.getElement("EMSX_SEQUENCE")

    for sq in options.sequence:
        print "SEQUENCE: %s" % (sq)
        sequences.appendValue(sq)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 500
...
```

Creating Basket from EMSX API Request

Creating a basket directly from the EMSX API requires the user to create a request from the service object of type *CreateOrder* with EMSX_BASKET_NAME specified. To route the basket, the user can use either *GroupRouteWithStrat* or *GroupRouteEx* and fill in the required fields before submitting the request. The EMSX_SEQUENCE numbers will be returned from *CreateOrder* feature and must be included in the *GroupRouteWithStrat* or *GroupRouteEx* requests. Any missing sequence number will result in order not being routed out by the basket.

The notion of basket defined here is to represent a list of orders sent to a single broker algorithm or destination. This feature is not for maintaining a state of basket at the buy-side end to send out to multiple different execution venues.

Trading on behalf of team members from TEAM_VIEW Request

Creating a route on behalf of the team member requires the user to create a request from the service object of type *RouteEx* and fill in the required fields before submitting the request.

The following is an example of the topic string for the request with a team name.

```
//blp/emapisvc/order;team=MY_TEAM_NAME?fields=EMSX_ASSIGNED_TRADER,EMSX_BASKET_NUM,EMSX_BSE_AVG_PRICE,EMSX_CFD_FLAG,API_SEQ_NUM
```

Inside *RouteEx*, there is a field EMSX_TRADER_UUID where the user can enter the order owner's UUID. Bloomberg will do the validation against the users privilege setup via EMT<GO> and EMBR<GO>. A user can be part of more than one team on the backend. When the user create the topic string and does not belong to a team or specify a wrong team, the user will get an error. If 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.

Multi-Leg Options using EMSX API

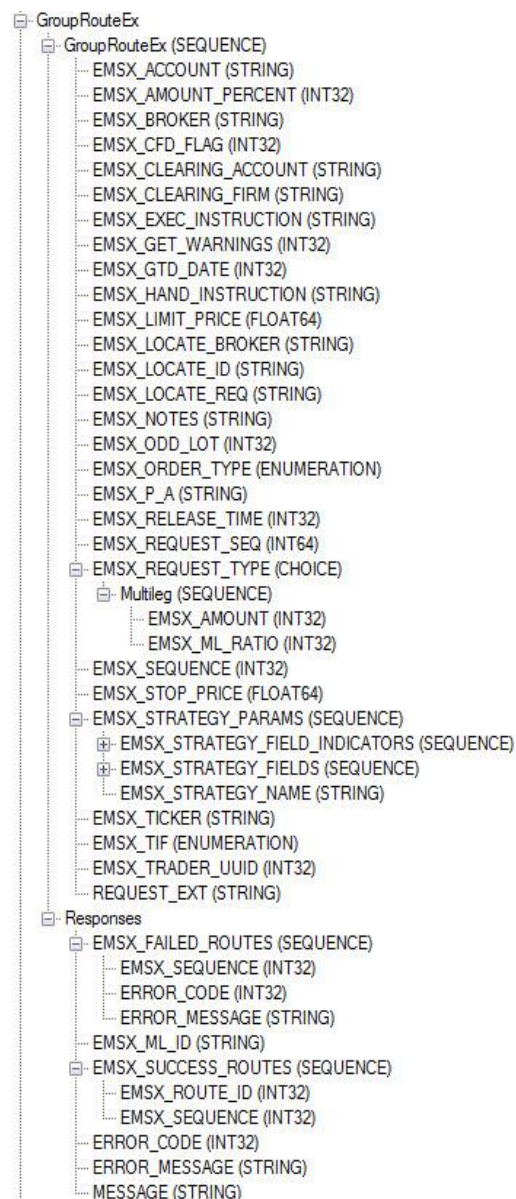
Multi-Leg Options can be traded using *GroupRouteEx* request. The first step is to create the options and if need be equities leg using *CreateOrder* request. Once this is completed, create a request object for *GroupRouteEx* and submit it to the session with all the fields necessary for the multi-leg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

Multileg 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, EMSX_ML_TOTAL_QUANTITY.



Bloomberg

Get All Field Meta Data Request

Create a request object for “GetAllFieldMetaData” and submit it to the session.

```
# EMSXGetAllFieldMetaDataRequest.py

import blpapi

GET_ALL_FIELD_METADATA = blpapi.Name("GetAllFieldMetaData")

d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
...
    request = service.createRequest("GetAllFieldMetaData")

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Get Field Meta Data Request

Create a request object for “GetFieldMetaData” and submit it to the session.

```
# EMSXGetFieldMetaDataRequest.py
...
    request = service.createRequest("GetFieldMetaData")

    fieldnames = request.getElement("EMSX_FIELD_NAMES")

    for f in options.fields:
        fieldnames.appendValue(f)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Get Teams Request

Get teams requires the user to create a request from the service object of type *GetTeams* and fill in the required fields before submitting the request.

Get teams returns the list of teams for the current user and the teams are customizable and therefore each client will utilize according to the operational flow of their particular firm.

```
# EMSXGetTeamsRequest.py
...
    request = service.createRequest("GetTeams")

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Get Brokers Request

Get brokers require the user to create a request from the service object of type *GetBrokers* and fill in the required fields before submitting the request.

Get brokers returns the list of brokers enabled for the current user for a specified security.

```
# EMSXGetBrokersRequest.py
...
    request = service.createRequest("GetBrokers")

    request.set("EMSX_TICKER", options.TICKER)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Get Broker Strategies Request

Get brokers require the user to create a request from the service object of type *GetBrokers* and fill in the required fields before submitting the request.

Get brokers returns the list of brokers enabled for the current user for a specified security.

```
# EMSXGetBrokerStrategiesRequest.py
...
    request = service.createRequest("GetBrokerStrategies")

    request.set("EMSX_BROKER", options.BROKER)
    request.set("EMSX_TICKER", options.TICKER)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Get Broker Strategy Information Request

Get brokers require the user to create a request from the service object of type *GetBrokers* and fill in the required fields before submitting the request.

Get brokers returns the list of brokers enabled for the current user for a specified security.

```
# EMSXGetBrokerStrategyInfoRequest.py
...
    request = service.createRequest("GetBrokerStrategyInfo")

    request.set("EMSX_BROKER", options.BROKER)
    request.set("EMSX_STRATEGY", options.STRATEGY)
    request.set("EMSX_TICKER", options.TICKER)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID )

    timeout = 5000
...
```

Order Information Request

Order Information request provides order details to the user. The user needs to create a request from the service object of type *OrderInfo* and fill in the required fields before submitting the request.

```
# EMSXOrderInfoRequest.py
...
    request = service.createRequest("OrderInfo")

    request.set("EMSX_SEQUENCE", options.SEQUENCE)
    request.set("EMSX_IS_AGGREGATED", options.AGGREGATED)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID)

    timeout = 5000
...
```

Route Information Request

Route Information request provides route details to the user. The user needs to create a request from the service object of type *RouteInfo* and fill in the required fields before submitting the request.

```
# EMSXRouteInfoRequest
...
    request = service.createRequest("RouteInfo")

    request.set("EMSX_SEQUENCE", options.SEQUENCE)
    request.set("EMSX_ROUTE_ID", options.ROUTEID)

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

    requestID = blpapi.CorrelationId(1)

    session.sendRequest(request, correlationId=requestID)

    timeout = 5000
...
```

Subscribing to Orders and Routes

Subscriptions provide a way of accessing and monitoring orders and routes (placements) in the user's blotter. The EMSX API provides access to all the orders and routes (placements) in a user's blotter via subscriptions. To establish a subscription the following information is required –

- a. Subscription Type – order or route
- b. Fields – The various EMSX fields that we are interested in

Once the subscription is established all the orders and/or routes (placements) in the user's blotter are returned via one or more BLP 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 (placements) will generate events that are passed along as they occur. These subscriptions can be synchronous or asynchronous.

Asynchronous Subscription

The synchronous example above can serve most needs but can prove to be a bottleneck in certain scenarios. To alleviate this, the BLP API offers asynchronous operations as well.

The following sample applications – *“EMSXSubscriptionOrdersAsync”* and *“EMSXSubscriptionsRoutesAsync”* illustrate this approach. This approach requires a callback method similar to below.

```
# EMSXSubscriptionsOrdersAsync.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")
...
d_service="//blp/emapisvc_beta"
d_host="localhost"
d_port=8194
...
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

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

            orderTopic = d_service + "/order?fields="
            orderTopic = orderTopic + "API_SEQ_NUM,"
            orderTopic = orderTopic + "EMSX_ACCOUNT,"
```

Bloomberg

```
orderTopic = orderTopic + "EMSX_AMOUNT,"
orderTopic = orderTopic + "EMSX_ARRIVAL_PRICE,"
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,"

print "Topic: %s" % orderTopic

subscriptionID = blpapi.CorrelationId(1)

subscriptions = blpapi.SubscriptionList()

subscriptions.add(topic=orderTopic,correlationId=subscriptionID)

session.subscribe(subscriptions)

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

```

EMSXSubscriptionsRoutesAsync

```
# EMSXSubscriptionsRoutesAsync.py
...
def processServiceStatusEvent(self,event,session):
    print "Processing SERVICE_STATUS event"

    for msg in event:

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

            routeTopic = d_service + "/route?fields="
            routeTopic = routeTopic + "EMSX_LAST_FILL_DATE,"
            routeTopic = routeTopic + "EMSX_LAST_FILL_TIME,"
            routeTopic = routeTopic + "EMSX_LAST_MARKET,"
            routeTopic = routeTopic + "EMSX_LAST_PRICE,"
            routeTopic = routeTopic + "EMSX_LAST_SHARES,"
            routeTopic = routeTopic + "EMSX_MISC_FEES,"

            print "Topic: %s" % routeTopic

            subscriptionID = blpapi.CorrelationId(1)

            subscriptions = blpapi.SubscriptionList()

            subscriptions.add(topic=routeTopic,correlationId=subscriptionID)

            session.subscribe(subscriptions)

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

```

Description of Subscription Message

Each order or route update is streamed to the client application as a sequence of messages. Each message in the sequence always contains the *header* seen below.

Subscription Message	
MSG_TYPE	'E' (This indicates that the message is an EMSX API message)
MSG_SUBTYPE	O=Order, R=Route
EVENT_STATUS	Can take values: HEARTBEAT INIT_PAINT NEW_ORDER_ROUTE UPD_ORDER_ROUTE DELETION_MESSAGE
API_SEQ_NUM	Sequence number and can be used to identify message gaps
EMSX_SEQUENCE	Order number in EMSX system
EMSX_ROUTE_ID	Route number (always 0 for order subscriptions)
EMSX_FILL_ID	Fill number of Route (<i>always 0 until we implement subscription of type fill</i>)

Description of Event Status Message

EVENT_STATUS	Message Type	Populated Fields
1	Heartbeat Message (HB_MESSAGE)	MSG_TYPE, API_SEQ_NUM and EVENT_STATUS
4	Initial paint and latest status for all user's orders and routes (INIT_PAINT)	All Subscription Fields
5	Unsupported broker algorithm replace NOT_INIT_PAINT	
6	New Order or Route (NEW_ORDER_ROUTE)	All Subscription Fields
7	Update for existing Orders and Routes. (UPD_ORDER_ROUTE)	Dynamic Fields that can change during Order and route lifespan
8	Order and Route deletion message (DELETION_MESSAGE)	Headers Only

Description of Order Status

Parent Order	Description of Status
NEW	Order has been added to the blotter; no routes have been created yet
SENT	Route has been sent to the broker but has not been acknowledged yet
WORKING	Route has been sent and acknowledged by the broker / Route has been partially filled/ Route has a cancel request pending or rejected. Associated child route status are WORKING, PARTFILLED, CXLREP, CXLREQ, CXLREJ, CXLRPQ, CXLRPJ, HOLD, REPPEN
PARTFILL	Route has been filled and the order has idle shares/ Route has been cancelled with partial fills. Associated child route status are FILLED, CANCEL
FILLED	All routes have been filled

EXPIRED	Derivatives orders convert to "Expired" status 4 hours after the trading session or at the start of the next session, and remain on the blotter for up to 48 hours. Associated child route status are CANCEL (unless the user touches the order i.e. CXL or MODR then it could be in WORKING)
---------	--

Description of Child Route Status

Child Route Status	Description of Status
SENT	Route has been sent to the broker but has not been acknowledged yet
WORKING	Route has been sent and acknowledged by the broker
REJECTED	Route has been rejected by the broker
PARTFILLED	Route has been partially filled
FILLED	Route has been filled
CXLREQ	Cancel Request is sent and is pending with the broker
CXLPEN	Cancel Pending
CXLREJ	Cancel Request is rejected by the broker
CANCEL	Cancel Request is accepted by the broker
CXLRPRQ	Cancel Replace Request is sent and is pending with the broker
CXLREP	Cancel Replace Request is accepted by the broker
CXLRPRJ	Cancel Replace Request is rejected by the broker
HOLD	Shares are committed to BMQ (TradeBook) or a dark pool
REPPEN	Replace Pending

AIM Specific Fields

Status			
QUEUED	ASSIGN	SUSPENDED	MODIFIED
MOD-REJ	MOD-PEND	COMPLETED	CAN-PEND
CAN-REJ	ORD-REJ	OA-SENT	PEND
A-SENT	ORD-PEND	STAGED	ALLO-PEND
ALLO-REJ	ALLOCATED	HOLD	CANCEL
DELETED	UNKNOWN	PENDING	CORRECTED
ROUTE-ERR	OMS PEND	DONE	COMPLETE

Bloomberg

Using Subscription and Request/Response Messages

The following is an example of *EMSXTrigger.java* to illustrate how a user can use a subscription data together with the request/response messages. The following example illustrates how to use subscription and request / response messages together with the following criteria.

- Single session
 - Single event handler
 - Every request must have correlation ID
 - Process **every** event
 - Subscriptions also need correlation ID is using *mktdata* and *emapisvc*
 - Identify the request from the correlation ID in order to know what to do with the results.
- The same logic applies for Subscription data.

```
/* EMSXTrigger.java */

package com.bloomberg.samples;

import java.util.ArrayList;
import java.util.Iterator;

import javax.print.DocFlavor.STRING;

import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Name;
import com.bloomberglp.blpapi.SessionOptions;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.Service;
import com.bloomberglp.blpapi.Request;
import com.bloomberglp.blpapi.Element;
import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.EventHandler;
import com.bloomberglp.blpapi.Subscription;
import com.bloomberglp.blpapi.SubscriptionList;

public class EMSXTrigger
{
    /* Compile with:-
    *   javac -cp c:\blp\API\APIv3\JavaAPI\v3.6.1.0\lib\blpapi3.jar
    *       com\bloomberg\samples\EMSXTrigger.java
    *
    * Run with:-
    *   java -cp .;c:\blp\API\APIv3\JavaAPI\v3.6.1.0\lib\blpapi3.jar
    *       com.bloomberg.samples.EMSXTrigger SELLAT=163.4 BUYAT=154.7 AMOUNT=1000
    *       TICKER="IBM US Equity"
    */

    private static final Name ORDER_ROUTE_FIELDS = new Name("OrderRouteFields");
```

Bloomberg

```
private static final Name ERROR_INFO = new Name("ErrorInfo");
private static final Name CREATE_ORDER_AND_ROUTE = new Name("CreateOrderAndRoute");

// ADMIN
private static final Name SLOW_CONSUMER_WARNING = new Name("SlowConsumerWarning");
private static final Name SLOW_CONSUMER_WARNING_CLEARED = new
    Name("SlowConsumerWarningCleared");

// SESSION_STATUS
private static final Name SESSION_STARTED = new Name("SessionStarted");
private static final Name SESSION_TERMINATED = new Name("SessionTerminated");
private static final Name SESSION_STARTUP_FAILURE = new Name("SessionStartupFailure");
private static final Name SESSION_CONNECTION_UP = new Name("SessionConnectionUp");
private static final Name SESSION_CONNECTION_DOWN = new Name("SessionConnectionDown");

// SERVICE_STATUS
private static final Name SERVICE_OPENED = new Name("ServiceOpened");
private static final Name SERVICE_OPEN_FAILURE = new Name("ServiceOpenFailure");

// SUBSCRIPTION_STATUS + SUBSCRIPTION_DATA
private static final Name SUBSCRIPTION_FAILURE = new Name("SubscriptionFailure");
private static final Name SUBSCRIPTION_STARTED = new Name("SubscriptionStarted");
private static final Name SUBSCRIPTION_TERMINATED = new Name("SubscriptionTerminated");

private static final Name SECURITY_DATA = new Name("securityData");
private static final Name SECURITY = new Name("security");
private static final Name FIELD_DATA = new Name("fieldData");

private Subscription emsx_order_sub;
private Subscription emsx_route_sub;
private SubscriptionList emsx_subscriptions = new SubscriptionList();
private CorrelationID order_sub_id;
private CorrelationID route_sub_id;

private Subscription mktdata_sub;
private SubscriptionList mktdata_subscriptions = new SubscriptionList();
private CorrelationID mktdata_sub_id;

private Request request;
private CorrelationID refdata_req_id;

private String d_host;
private int d_port;

private static String service_mktdata = "//blp/mktdata";
private static String service_emsx = "//blp/emapisvc_beta";
private static String service_refdata = "//blp/refdata";

private Service refdata;
private Service emapisvc;

private double sellat = 0;
private double buyat = 0;
private int amount = 0;
private String ticker = "";

private int emsx_req_id = 100;
```

Bloomberg

```
public class IDSequenceMap {
    public CorrelationID requestID;
    public int sequenceNo;
    public int routeID;
}

private ArrayList<IDSequenceMap> idMapping = new ArrayList<IDSequenceMap>();

public static void main(String[] args) throws java.lang.Exception
{
    System.out.println("Bloomberg - EMSX API Example - EMSXSubscriptionsOrdersAsync");
    System.out.println("Press ENTER at anytime to quit");

    EMSXTrigger example = new EMSXTrigger();
    example.run(args);

    System.in.read();
}

public EMSXTrigger()
{
    // Define the service required, in this case the beta service,
    // and the values to be used by the SessionOptions object
    // to identify IP/port of the back-end process.

    d_host = "localhost";
    d_port = 8194;
}

private void run(String[] args) throws Exception
{
    if (!parseCommandLine(args)) return;

    SessionOptions d_sessionOptions = new SessionOptions();
    d_sessionOptions.setServerHost(d_host);
    d_sessionOptions.setServerPort(d_port);

    Session session = new Session(d_sessionOptions, new EMSXEventHandler());

    session.startAsync();
}

class EMSXEventHandler implements EventHandler
{
    public void processEvent(Event evt, Session session)
    {
        try
        {
            switch (evt.eventType().intValue())
            {
                case Event.EventType.Constants.ADMIN:
                    processAdminEvent(evt, session);
                    break;
                case Event.EventType.Constants.SESSION_STATUS:
```

```
        processSessionEvent(evt, session);
        break;
    case Event.EventType.Constants.SERVICE_STATUS:
        processServiceEvent(evt, session);
        break;
    case Event.EventType.Constants.SUBSCRIPTION_DATA:
        processSubscriptionDataEvent(evt, session);
        break;
    case Event.EventType.Constants.SUBSCRIPTION_STATUS:
        processSubscriptionStatusEvent(evt, session);
        break;
    case Event.EventType.Constants.RESPONSE:
    case Event.EventType.Constants.PARTIAL_RESPONSE:
        processResponseEvent(evt, session);
        break;
    default:
        processMiscEvent(evt, session);
        break;
    }
}
catch (Exception e)
{
    System.err.println("Error: " + e.toString());
}
}

private void processAdminEvent(Event evt, Session session)
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {
        Message msg = msgIter.next();

        if (msg.messageType().equals(SLOW_CONSUMER_WARNING))
        {
            System.err.println("Warning: Entered Slow Consumer status");
        }
        else if (msg.messageType().equals(SLOW_CONSUMER_WARNING_CLEARED))
        {
            System.err.println("Slow consumer status cleared");
        }
    }
}

private void processSessionEvent(Event evt, Session session) throws Exception
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {
        Message msg = msgIter.next();

        if (msg.messageType().equals(SESSION_STARTED))
        {
            System.out.println("Session started...");
            session.openServiceAsync(service_refdata);
            session.openServiceAsync(service_mktdata);
            session.openServiceAsync(service_emsx);
        }
    }
}
```

```
    }
    else if (msg.messageType().equals(SESSION_STARTUP_FAILURE))
    {
        System.err.println("Error: Session startup failed");
    }
    else if (msg.messageType().equals(SESSION_TERMINATED))
    {
        System.err.println("Error: Session has been terminated");
    }
    else if (msg.messageType().equals(SESSION_CONNECTION_UP))
    {
        System.out.println("Session connection is up");
    }
    else if (msg.messageType().equals(SESSION_CONNECTION_DOWN))
    {
        System.err.println("Error: Session connection is down");
    }
}

}

private void processServiceEvent(Event evt, Session session) throws Exception
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {
        Message msg = msgIter.next();

        // Identify which service this message belongs to...
        String svc = msg.getElementAsString("serviceName");

        if (msg.messageType().equals(SERVICE_OPENED))
        {
            System.out.println("Service opened [" + svc + "...");

            if (svc == service_refdata)
            {
                // Static data request to retrieve the ticker description
                // (PARSEKYABLE_DES_SOURCE)
                refdata = session.getService(service_refdata);

                request = refdata.createRequest("ReferenceDataRequest");
                Element securities = request.getElement("securities");
                securities.appendValue(ticker);
                Element fields = request.getElement("fields");
                fields.appendValue("PARSEKYABLE_DES_SOURCE");

                System.out.println("Sending Request: " + request);

                refdata_req_id = new CorrelationID(3);

                session.sendRequest(request, refdata_req_id);
            }
            else if (svc == service_mktdata)
            {
                // Create subscription for LAST_PRICE

                mktdata_sub_id = new CorrelationID(4);
            }
        }
    }
}
```

```
mktdata_sub = new Subscription(ticker, "LAST_PRICE",
    mktdata_sub_id);
System.out.println("Market Data Topic: " + mktdata_sub);
mktdata_subscriptions.add(mktdata_sub);

try
{
    session.subscribe(mktdata_subscriptions);
}
catch (Exception ex)
{
    System.err.println("Failed to create Market Data subscription:
        " + ex.getMessage());
}

}
else if (svc == service_emsx)
{

    emapisvc = session.getService(service_emsx);

    // Create the topic string for the subscription. Here, we are
    // subscribing
    // to every available order field, however, you can subscribe to
    // only the fields
    // required for your application.
    String orderTopic = service_emsx + "/order?fields=";

    orderTopic = orderTopic + "API_SEQ_NUM,";
    orderTopic = orderTopic + "EMSX_ACCOUNT,";
    orderTopic = orderTopic + "EMSX_AMOUNT,";
    orderTopic = orderTopic + "EMSX_ARRIVAL_PRICE,";
    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,";
    orderTopic = orderTopic + "EMSX_DAY_FILL,";
    orderTopic = orderTopic + "EMSX_DIR_BROKER_FLAG,";
    orderTopic = orderTopic + "EMSX_EXCHANGE,";
    orderTopic = orderTopic + "EMSX_EXCHANGE_DESTINATION,";
    orderTopic = orderTopic + "EMSX_EXEC_INSTRUCTION,";
    orderTopic = orderTopic + "EMSX_FILL_ID,";
    orderTopic = orderTopic + "EMSX_FILLED,";
    orderTopic = orderTopic + "EMSX_GTD_DATE,";
    orderTopic = orderTopic + "EMSX_HAND_INSTRUCTION,";
    orderTopic = orderTopic + "EMSX_IDLE_AMOUNT,";
    orderTopic = orderTopic + "EMSX_INVESTOR_ID,";
    orderTopic = orderTopic + "EMSX_ISIN,";
```

```
orderTopic = orderTopic + "EMSX_LIMIT_PRICE,";
orderTopic = orderTopic + "EMSX_NOTES,";
orderTopic = orderTopic + "EMSX_NSE_AVG_PRICE,";
orderTopic = orderTopic + "EMSX_NSE_FILLED,";
orderTopic = orderTopic + "EMSX_ORD_REF_ID,";
orderTopic = orderTopic + "EMSX_ORDER_TYPE,";
orderTopic = orderTopic + "EMSX_ORIGINATE_TRADER,";
orderTopic = orderTopic + "EMSX_ORIGINATE_TRADER_FIRM,";
orderTopic = orderTopic + "EMSX_PERCENT_REMAIN,";
orderTopic = orderTopic + "EMSX_PM_UUID,";
orderTopic = orderTopic + "EMSX_PORT_MGR,";
orderTopic = orderTopic + "EMSX_PORT_NAME,";
orderTopic = orderTopic + "EMSX_PORT_NUM,";
orderTopic = orderTopic + "EMSX_POSITION,";
orderTopic = orderTopic + "EMSX_PRINCIPAL,";
orderTopic = orderTopic + "EMSX_PRODUCT,";
orderTopic = orderTopic + "EMSX_QUEUED_DATE,";
orderTopic = orderTopic + "EMSX_QUEUED_TIME,";
orderTopic = orderTopic + "EMSX_REASON_CODE,";
orderTopic = orderTopic + "EMSX_REASON_DESC,";
orderTopic = orderTopic + "EMSX_REMAIN_BALANCE,";
orderTopic = orderTopic + "EMSX_ROUTE_ID,";
orderTopic = orderTopic + "EMSX_ROUTE_PRICE,";
orderTopic = orderTopic + "EMSX_SEC_NAME,";
orderTopic = orderTopic + "EMSX_SEDOL,";
orderTopic = orderTopic + "EMSX_SEQUENCE,";
orderTopic = orderTopic + "EMSX_SETTLE_AMOUNT,";
orderTopic = orderTopic + "EMSX_SETTLE_DATE,";
orderTopic = orderTopic + "EMSX_SIDE,";
orderTopic = orderTopic + "EMSX_START_AMOUNT,";
orderTopic = orderTopic + "EMSX_STATUS,";
orderTopic = orderTopic + "EMSX_STEP_OUT_BROKER,";
orderTopic = orderTopic + "EMSX_STOP_PRICE,";
orderTopic = orderTopic + "EMSX_STRATEGY_END_TIME,";
orderTopic = orderTopic + "EMSX_STRATEGY_PART_RATE1,";
orderTopic = orderTopic + "EMSX_STRATEGY_PART_RATE2,";
orderTopic = orderTopic + "EMSX_STRATEGY_START_TIME,";
orderTopic = orderTopic + "EMSX_STRATEGY_STYLE,";
orderTopic = orderTopic + "EMSX_STRATEGY_TYPE,";
orderTopic = orderTopic + "EMSX_TICKER,";
orderTopic = orderTopic + "EMSX_TIF,";
orderTopic = orderTopic + "EMSX_TIME_STAMP,";
orderTopic = orderTopic + "EMSX_TRAD_UUID,";
orderTopic = orderTopic + "EMSX_TRADE_DESK,";
orderTopic = orderTopic + "EMSX_TRADER,";
orderTopic = orderTopic + "EMSX_TRADER_NOTES,";
orderTopic = orderTopic + "EMSX_TS_ORDNUM,";
orderTopic = orderTopic + "EMSX_TYPE,";
orderTopic = orderTopic + "EMSX_UNDERLYING_TICKER,";
orderTopic = orderTopic + "EMSX_USER_COMM_AMOUNT,";
orderTopic = orderTopic + "EMSX_USER_COMM_RATE,";
orderTopic = orderTopic + "EMSX_USER_FEES,";
orderTopic = orderTopic + "EMSX_USER_NET_MONEY,";
orderTopic = orderTopic + "EMSX_WORK_PRICE,";
orderTopic = orderTopic + "EMSX_WORKING,";
orderTopic = orderTopic + "EMSX_YELLOW_KEY";
```

```
// We define a correlation ID that allows us to identify the
```



```
// original
// request when we examine the responses. This is useful in
// situations where
// the same event handler is used the process messages for the
// Order and Route
// subscriptions, as well as request/response requests.
order_sub_id = new CorrelationID(1);

emsx_order_sub= new Subscription(orderTopic, order_sub_id);
System.out.println("Order Topic: " + orderTopic);
emsx_subscriptions.add(emsx_order_sub);

String routeTopic = service_emsx + "/route?fields=";

routeTopic = routeTopic + "API_SEQ_NUM,";
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,";
routeTopic = routeTopic + "EMSX_COMM_DIFF_FLAG,";
routeTopic = routeTopic + "EMSX_COMM_RATE,";
routeTopic = routeTopic + "EMSX_CURRENCY_PAIR,";
routeTopic = routeTopic + "EMSX_CUSTOM_ACCOUNT,";
routeTopic = routeTopic + "EMSX_DAY_AVG_PRICE,";
routeTopic = routeTopic + "EMSX_DAY_FILL,";
routeTopic = routeTopic + "EMSX_EXCHANGE_DESTINATION,";
routeTopic = routeTopic + "EMSX_EXEC_INSTRUCTION,";
routeTopic = routeTopic + "EMSX_EXECUTE_BROKER,";
routeTopic = routeTopic + "EMSX_FILL_ID,";
routeTopic = routeTopic + "EMSX_FILLED,";
routeTopic = routeTopic + "EMSX_GTD_DATE,";
routeTopic = routeTopic + "EMSX_HAND_INSTRUCTION,";
routeTopic = routeTopic + "EMSX_IS_MANUAL_ROUTE,";
routeTopic = routeTopic + "EMSX_LAST_FILL_DATE,";
routeTopic = routeTopic + "EMSX_LAST_FILL_TIME,";
routeTopic = routeTopic + "EMSX_LAST_MARKET,";
routeTopic = routeTopic + "EMSX_LAST_PRICE,";
routeTopic = routeTopic + "EMSX_LAST_SHARES,";
routeTopic = routeTopic + "EMSX_LIMIT_PRICE,";
routeTopic = routeTopic + "EMSX_MISC_FEES,";
routeTopic = routeTopic + "EMSX_ML_LEG_QUANTITY,";
routeTopic = routeTopic + "EMSX_ML_NUM_LEGS,";
routeTopic = routeTopic + "EMSX_ML_PERCENT_FILLED,";
routeTopic = routeTopic + "EMSX_ML_RATIO,";
routeTopic = routeTopic + "EMSX_ML_REMAIN_BALANCE,";
routeTopic = routeTopic + "EMSX_ML_STRATEGY,";
routeTopic = routeTopic + "EMSX_ML_TOTAL_QUANTITY,";
routeTopic = routeTopic + "EMSX_NOTES,";
routeTopic = routeTopic + "EMSX_NSE_AVG_PRICE,";
routeTopic = routeTopic + "EMSX_NSE_FILLED,";
routeTopic = routeTopic + "EMSX_ORDER_TYPE,";
routeTopic = routeTopic + "EMSX_P_A,";
routeTopic = routeTopic + "EMSX_PERCENT_REMAIN,";
routeTopic = routeTopic + "EMSX_PRINCIPAL,";
routeTopic = routeTopic + "EMSX_QUEUED_DATE,";
```

```
routeTopic = routeTopic + "EMSX_QUEUED_TIME,";
routeTopic = routeTopic + "EMSX_REASON_CODE,";
routeTopic = routeTopic + "EMSX_REASON_DESC,";
routeTopic = routeTopic + "EMSX_REMAIN_BALANCE,";
routeTopic = routeTopic + "EMSX_ROUTE_CREATE_DATE,";
routeTopic = routeTopic + "EMSX_ROUTE_CREATE_TIME,";
routeTopic = routeTopic + "EMSX_ROUTE_ID,";
routeTopic = routeTopic + "EMSX_ROUTE_LAST_UPDATE_TIME,";
routeTopic = routeTopic + "EMSX_ROUTE_PRICE,";
routeTopic = routeTopic + "EMSX_SEQUENCE,";
routeTopic = routeTopic + "EMSX_SETTLE_AMOUNT,";
routeTopic = routeTopic + "EMSX_SETTLE_DATE,";
routeTopic = routeTopic + "EMSX_STATUS,";
routeTopic = routeTopic + "EMSX_STOP_PRICE,";
routeTopic = routeTopic + "EMSX_STRATEGY_END_TIME,";
routeTopic = routeTopic + "EMSX_STRATEGY_PART_RATE1,";
routeTopic = routeTopic + "EMSX_STRATEGY_PART_RATE2,";
routeTopic = routeTopic + "EMSX_STRATEGY_START_TIME,";
routeTopic = routeTopic + "EMSX_STRATEGY_STYLE,";
routeTopic = routeTopic + "EMSX_STRATEGY_TYPE,";
routeTopic = routeTopic + "EMSX_TIF,";
routeTopic = routeTopic + "EMSX_TIME_STAMP,";
routeTopic = routeTopic + "EMSX_TYPE,";
routeTopic = routeTopic + "EMSX_URGENCY_LEVEL,";
routeTopic = routeTopic + "EMSX_USER_COMM_AMOUNT,";
routeTopic = routeTopic + "EMSX_USER_COMM_RATE,";
routeTopic = routeTopic + "EMSX_USER_FEES,";
routeTopic = routeTopic + "EMSX_USER_NET_MONEY,";
routeTopic = routeTopic + "EMSX_WORKING";

route_sub_id = new CorrelationID(2);

emsx_route_sub = new Subscription(routeTopic, route_sub_id);
System.out.println("Route Topic: " + routeTopic);
emsx_subscriptions.add(emsx_route_sub);

try
{
    session.subscribe(emsx_subscriptions);
}
catch (Exception ex)
{
    System.err.println("Failed to create EMSX subscription: " +
        ex.getMessage());
}
}
else if (msg.messageType().equals(SERVICE_OPEN_FAILURE))
{
    System.err.println("Error: Service failed to open [" + svc + "]);
}
}

private void processSubscriptionStatusEvent(Event evt, Session session)
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
```

Bloomberg

```
while (msgIter.hasNext()) {
    Message msg = msgIter.next();

    if (msg.messageType().equals(SUBSCRIPTION_STARTED))
    {
        if (msg.correlationID() == mktdata_sub_id)
        {
            System.out.println("Subscription started [" + service_mktdata +
                "]);");
        }
        else if (msg.correlationID() == order_sub_id)
        {
            System.out.println("Subscription started [" + service_emsx + " -
                Orders]);");
        }
        else if (msg.correlationID() == route_sub_id)
        {
            System.out.println("Subscription started [" + service_emsx + " -
                Routes]);");
        }
    }
    else if (msg.messageType().equals(SUBSCRIPTION_FAILURE))
    {
        System.err.println("Error: Subscription failed");
        System.out.println("MESSAGE: " + msg);
    }
    else if (msg.messageType().equals(SUBSCRIPTION_TERMINATED))
    {
        System.err.println("Error: Subscription terminated");
        System.out.println("MESSAGE: " + msg);
    }
}

private void processSubscriptionDataEvent(Event evt, Session session) throws
Exception
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {

        Message msg = msgIter.next();

        if (msg.correlationID() == mktdata_sub_id)
        {
            System.out.println("Subscription data for MktData");

            //System.out.println(msg);
            Double last_price = msg.hasElement("LAST_PRICE") ?
                msg.getElementAsFloat64("LAST_PRICE") : 0;
            if (last_price > 0)
            {
                System.out.println("MktData: LAST_PRICE = " +
                    String.format("%.4f",last_price));

                if (last_price < buyat || last_price > sellat)
                {
```

```
Request request =
    emapisvc.createRequest("CreateOrderAndRoute");

request.set("EMSX_TICKER", ticker);
request.set("EMSX_AMOUNT", amount);
request.set("EMSX_ORDER_TYPE", "MKT");
request.set("EMSX_BROKER", "BB");
request.set("EMSX_TIF", "DAY");
request.set("EMSX_HAND_INSTRUCTION", "ANY");

if(last_price < buyat) request.set("EMSX_SIDE", "BUY");
else request.set("EMSX_SIDE", "SELL");

CorrelationID requestID = new CorrelationID(emsx_req_id++);

IDSequenceMap map = new IDSequenceMap();
map.requestID = requestID;

idMapping.add(map);

session.sendRequest(request, requestID);
    }
}
}
else if (msg.correlationID() == order_sub_id || msg.correlationID() ==
route_sub_id)
{
    System.out.println("Subscription data for EMSX Orders");

    int event_status = msg.getElementAsInt32("EVENT_STATUS");

    if (event_status == 1)
    {
        System.out.println("EMSX Subscription Heartbeat...");
    }
    else
    {
        String msg_sub_type = msg.hasElement("MSG_SUB_TYPE") ?
            msg.getElementAsString("MSG_SUB_TYPE") : "";

        if (msg_sub_type == "O")
        {
            long api_seq_num = msg.hasElement("API_SEQ_NUM") ?
                msg.getElementAsInt64("API_SEQ_NUM") : 0;
            String emsx_account = msg.hasElement("EMSX_ACCOUNT") ?
                msg.getElementAsString("EMSX_ACCOUNT") : "";
            int emsx_amount = msg.hasElement("EMSX_AMOUNT") ?
                msg.getElementAsInt32("EMSX_AMOUNT") : 0;
            double emsx_arrival_price =
                msg.hasElement("EMSX_ARRIVAL_PRICE") ?
                msg.getElementAsFloat64("EMSX_ARRIVAL_PRICE") : 0;
            String emsx_asset_class = msg.hasElement("EMSX_ASSET_CLASS") ?
                msg.getElementAsString("EMSX_ASSET_CLASS") : "";
            String emsx_assigned_trader =
                msg.hasElement("EMSX_ASSIGNED_TRADER") ?
                msg.getElementAsString("EMSX_ASSIGNED_TRADER") : "";
            double emsx_avg_price = msg.hasElement("EMSX_AVG_PRICE") ?
                msg.getElementAsFloat64("EMSX_AVG_PRICE") : 0;
        }
    }
}
```

```
String emsx_basket_name = msg.hasElement("EMSX_BASKET_NAME") ?
    msg.getElementAsString("EMSX_BASKET_NAME") : "";
int emsx_basket_num = msg.hasElement("EMSX_BASKET_NUM") ?
    msg.getElementAsInt32("EMSX_BASKET_NUM") : 0;
String emsx_broker = msg.hasElement("EMSX_BROKER") ?
    msg.getElementAsString("EMSX_BROKER") : "";
double emsx_broker_comm = msg.hasElement("EMSX_BROKER_COMM") ?
    msg.getElementAsFloat64("EMSX_BROKER_COMM") : 0;
double emsx_bse_avg_price =
    msg.hasElement("EMSX_BSE_AVG_PRICE") ?
    msg.getElementAsFloat64("EMSX_BSE_AVG_PRICE") : 0;
int emsx_bse_filled = msg.hasElement("EMSX_BSE_FILLED") ?
    msg.getElementAsInt32("EMSX_BSE_FILLED") : 0;
String emsx_cfd_flag = msg.hasElement("EMSX_CFD_FLAG") ?
    msg.getElementAsString("EMSX_CFD_FLAG") : "";
String emsx_comm_diff_flag =
    msg.hasElement("EMSX_COMM_DIFF_FLAG") ?
    msg.getElementAsString("EMSX_COMM_DIFF_FLAG") : "";
double emsx_comm_rate = msg.hasElement("EMSX_COMM_RATE") ?
    msg.getElementAsFloat64("EMSX_COMM_RATE") : 0;
String emsx_currency_pair =
    msg.hasElement("EMSX_CURRENCY_PAIR") ?
    msg.getElementAsString("EMSX_CURRENCY_PAIR") : "";
int emsx_date = msg.hasElement("EMSX_DATE") ?
    msg.getElementAsInt32("EMSX_DATE") : 0;
double emsx_day_avg_price =
    msg.hasElement("EMSX_DAY_AVG_PRICE") ?
    msg.getElementAsFloat64("EMSX_DAY_AVG_PRICE") : 0;
int emsx_day_fill = msg.hasElement("EMSX_DAY_FILL") ?
    msg.getElementAsInt32("EMSX_DAY_FILL") : 0;
String emsx_dir_broker_flag =
    msg.hasElement("EMSX_DIR_BROKER_FLAG") ?
    msg.getElementAsString("EMSX_DIR_BROKER_FLAG") : "";
String emsx_exchange = msg.hasElement("EMSX_EXCHANGE") ?
    msg.getElementAsString("EMSX_EXCHANGE") : "";
String emsx_exchange_destination =
    msg.hasElement("EMSX_EXCHANGE_DESTINATION") ?
    msg.getElementAsString("EMSX_EXCHANGE_DESTINATION") : "";
String emsx_exec_instruction =
    msg.hasElement("EMSX_EXEC_INSTRUCTION") ?
    msg.getElementAsString("EMSX_EXEC_INSTRUCTION") : "";
int emsx_fill_id = msg.hasElement("EMSX_FILL_ID") ?
    msg.getElementAsInt32("EMSX_FILL_ID") : 0;
int emsx_filled = msg.hasElement("EMSX_FILLED") ?
    msg.getElementAsInt32("EMSX_FILLED") : 0;
int emsx_gtd_date = msg.hasElement("EMSX_GTD_DATE") ?
    msg.getElementAsInt32("EMSX_GTD_DATE") : 0;
String emsx_hand_instruction =
    msg.hasElement("EMSX_HAND_INSTRUCTION") ?
    msg.getElementAsString("EMSX_HAND_INSTRUCTION") : "";
int emsx_idle_amount = msg.hasElement("EMSX_IDLE_AMOUNT") ?
    msg.getElementAsInt32("EMSX_IDLE_AMOUNT") : 0;
String emsx_investor_id = msg.hasElement("EMSX_INVESTOR_ID") ?
    msg.getElementAsString("EMSX_INVESTOR_ID") : "";
String emsx_isin = msg.hasElement("EMSX_ISIN") ?
    msg.getElementAsString("EMSX_ISIN") : "";
double emsx_limit_price = msg.hasElement("EMSX_LIMIT_PRICE") ?
    msg.getElementAsFloat64("EMSX_LIMIT_PRICE") : 0;
String emsx_notes = msg.hasElement("EMSX_NOTES") ?
```

```
msg.getElementAsString("EMSX_NOTES") : "";
double emsx_nse_avg_price =
msg.hasElement("EMSX_NSE_AVG_PRICE") ?
msg.getElementAsFloat64("EMSX_NSE_AVG_PRICE") : 0;
int emsx_nse_filled = msg.hasElement("EMSX_NSE_FILLED") ?
msg.getElementAsInt32("EMSX_NSE_FILLED") : 0;
String emsx_ord_ref_id = msg.hasElement("EMSX_ORD_REF_ID") ?
msg.getElementAsString("EMSX_ORD_REF_ID") : "";
String emsx_order_type = msg.hasElement("EMSX_ORDER_TYPE") ?
msg.getElementAsString("EMSX_ORDER_TYPE") : "";
String emsx_originate_trader =
msg.hasElement("EMSX_ORIGINATE_TRADER") ?
msg.getElementAsString("EMSX_ORIGINATE_TRADER") : "";
String emsx_originate_trader_firm =
msg.hasElement("EMSX_ORIGINATE_TRADER_FIRM") ?
msg.getElementAsString("EMSX_ORIGINATE_TRADER_FIRM") : "";
double emsx_percent_remain =
msg.hasElement("EMSX_PERCENT_REMAIN") ?
msg.getElementAsFloat64("EMSX_PERCENT_REMAIN") : 0;
int emsx_pm_uuid = msg.hasElement("EMSX_PM_UUID") ?
msg.getElementAsInt32("EMSX_PM_UUID") : 0;
String emsx_port_mgr = msg.hasElement("EMSX_PORT_MGR") ?
msg.getElementAsString("EMSX_PORT_MGR") : "";
String emsx_port_name = msg.hasElement("EMSX_PORT_NAME") ?
msg.getElementAsString("EMSX_PORT_NAME") : "";
int emsx_port_num = msg.hasElement("EMSX_PORT_NUM") ?
msg.getElementAsInt32("EMSX_PORT_NUM") : 0;
String emsx_position = msg.hasElement("EMSX_POSITION") ?
msg.getElementAsString("EMSX_POSITION") : "";
double emsx_principle = msg.hasElement("EMSX_PRINCIPAL") ?
msg.getElementAsFloat64("EMSX_PRINCIPAL") : 0;
String emsx_product = msg.hasElement("EMSX_PRODUCT") ?
msg.getElementAsString("EMSX_PRODUCT") : "";
int emsx_queued_date = msg.hasElement("EMSX_QUEUED_DATE") ?
msg.getElementAsInt32("EMSX_QUEUED_DATE") : 0;
int emsx_queued_time = msg.hasElement("EMSX_QUEUED_TIME") ?
msg.getElementAsInt32("EMSX_QUEUED_TIME") : 0;
String emsx_reason_code = msg.hasElement("EMSX_REASON_CODE") ?
msg.getElementAsString("EMSX_REASON_CODE") : "";
String emsx_reason_desc = msg.hasElement("EMSX_REASON_DESC") ?
msg.getElementAsString("EMSX_REASON_DESC") : "";
double emsx_remain_balance =
msg.hasElement("EMSX_REMAIN_BALANCE") ?
msg.getElementAsFloat64("EMSX_REMAIN_BALANCE") : 0;
int emsx_route_id = msg.hasElement("EMSX_ROUTE_ID") ?
msg.getElementAsInt32("EMSX_ROUTE_ID") : 0;
double emsx_route_price = msg.hasElement("EMSX_ROUTE_PRICE") ?
msg.getElementAsFloat64("EMSX_ROUTE_PRICE") : 0;
String emsx_sec_name = msg.hasElement("EMSX_SEC_NAME") ?
msg.getElementAsString("EMSX_SEC_NAME") : "";
String emsx_sedol = msg.hasElement("EMSX_SEDOL") ?
msg.getElementAsString("EMSX_SEDOL") : "";
int emsx_sequence = msg.hasElement("EMSX_SEQUENCE") ?
msg.getElementAsInt32("EMSX_SEQUENCE") : 0;
double emsx_settle_amount =
msg.hasElement("EMSX_SETTLE_AMOUNT") ?
msg.getElementAsFloat64("EMSX_SETTLE_AMOUNT") : 0;
int emsx_settle_date = msg.hasElement("EMSX_SETTLE_DATE") ?
msg.getElementAsInt32("EMSX_SETTLE_DATE") : 0;
```

```
String emsx_side = msg.hasElement("EMSX_SIDE") ?
    msg.getElementAsString("EMSX_SIDE") : "";
int emsx_start_amount = msg.hasElement("EMSX_START_AMOUNT") ?
    msg.getElementAsInt32("EMSX_START_AMOUNT") : 0;
String emsx_status = msg.hasElement("EMSX_STATUS") ?
    msg.getElementAsString("EMSX_STATUS") : "";
String emsx_step_out_broker =
    msg.hasElement("EMSX_STEP_OUT_BROKER") ?
    msg.getElementAsString("EMSX_STEP_OUT_BROKER") : "";
double emsx_stop_price = msg.hasElement("EMSX_STOP_PRICE") ?
    msg.getElementAsFloat64("EMSX_STOP_PRICE") : 0;
int emsx_strategy_end_time =
    msg.hasElement("EMSX_STRATEGY_END_TIME") ?
    msg.getElementAsInt32("EMSX_STRATEGY_END_TIME") : 0;
double emsx_strategy_part_rate1 =
    msg.hasElement("EMSX_STRATEGY_PART_RATE1") ?
    msg.getElementAsFloat64("EMSX_STRATEGY_PART_RATE1") : 0;
double emsx_strategy_part_rate2 =
    msg.hasElement("EMSX_STRATEGY_PART_RATE2") ?
    msg.getElementAsFloat64("EMSX_STRATEGY_PART_RATE2") : 0;
int emsx_strategy_start_time =
    msg.hasElement("EMSX_STRATEGY_START_TIME") ?
    msg.getElementAsInt32("EMSX_STRATEGY_START_TIME") : 0;
String emsx_strategy_style =
    msg.hasElement("EMSX_STRATEGY_STYLE") ?
    msg.getElementAsString("EMSX_STRATEGY_STYLE") : "";
String emsx_strategy_type =
    msg.hasElement("EMSX_STRATEGY_TYPE") ?
    msg.getElementAsString("EMSX_STRATEGY_TYPE") : "";
String emsx_ticker = msg.hasElement("EMSX_TICKER") ?
    msg.getElementAsString("EMSX_TICKER") : "";
String emsx_tif = msg.hasElement("EMSX_TIF") ?
    msg.getElementAsString("EMSX_TIF") : "";
int emsx_time_stamp = msg.hasElement("EMSX_TIME_STAMP") ?
    msg.getElementAsInt32("EMSX_TIME_STAMP") : 0;
int emsx_trad_uuid = msg.hasElement("EMSX_TRAD_UUID") ?
    msg.getElementAsInt32("EMSX_TRAD_UUID") : 0;
String emsx_trade_desk = msg.hasElement("EMSX_TRADE_DESK") ?
    msg.getElementAsString("EMSX_TRADE_DESK") : "";
String emsx_trader = msg.hasElement("EMSX_TRADER") ?
    msg.getElementAsString("EMSX_TRADER") : "";
String emsx_trader_notes =
    msg.hasElement("EMSX_TRADER_NOTES") ?
    msg.getElementAsString("EMSX_TRADER_NOTES") : "";
int emsx_ts_ordnum = msg.hasElement("EMSX_TS_ORDNUM") ?
    msg.getElementAsInt32("EMSX_TS_ORDNUM") : 0;
String emsx_type = msg.hasElement("EMSX_TYPE") ?
    msg.getElementAsString("EMSX_TYPE") : "";
String emsx_underlying_ticker =
    msg.hasElement("EMSX_UNDERLYING_TICKER") ?
    msg.getElementAsString("EMSX_UNDERLYING_TICKER") : "";
double emsx_user_comm_amount =
    msg.hasElement("EMSX_USER_COMM_AMOUNT") ?
    msg.getElementAsFloat64("EMSX_USER_COMM_AMOUNT") : 0;
double emsx_user_comm_rate =
    msg.hasElement("EMSX_USER_COMM_RATE") ?
    msg.getElementAsFloat64("EMSX_USER_COMM_RATE") : 0;
double emsx_user_fees = msg.hasElement("EMSX_USER_FEES") ?
    msg.getElementAsFloat64("EMSX_USER_FEES") : 0;
```



```
double emsx_user_net_money =
    msg.hasElement("EMSX_USER_NET_MONEY") ?
    msg.getElementAsFloat64("EMSX_USER_NET_MONEY") : 0;
double emsx_user_work_price =
    msg.hasElement("EMSX_WORK_PRICE") ?
    msg.getElementAsFloat64("EMSX_WORK_PRICE") : 0;
int emsx_working = msg.hasElement("EMSX_WORKING") ?
    msg.getElementAsInt32("EMSX_WORKING") : 0;
String emsx_yellow_key = msg.hasElement("EMSX_YELLOW_KEY") ?
    msg.getElementAsString("EMSX_YELLOW_KEY") : "";

System.out.println("ORDER MESSAGE: CorrelationID(" +
    msg.correlationID() + ") EMSX_SEQUENCE: " +
    Integer.toString(emsx_sequence) + "\t EVENT_STATUS: " +
    event_status + "\t EMSX_STATUS: " + emsx_status);
}
else if (msg_sub_type == "R")
{
    long api_seq_num = msg.hasElement("API_SEQ_NUM") ?
        msg.getElementAsInt64("API_SEQ_NUM") : 0;
    int emsx_amount = msg.hasElement("EMSX_AMOUNT") ?
        msg.getElementAsInt32("EMSX_AMOUNT") : 0;
    double emsx_avg_price = msg.hasElement("EMSX_AVG_PRICE") ?
        msg.getElementAsFloat64("EMSX_AVG_PRICE") : 0;
    String emsx_broker = msg.hasElement("EMSX_BROKER") ?
        msg.getElementAsString("EMSX_BROKER") : "";
    double emsx_broker_comm = msg.hasElement("EMSX_BROKER_COMM") ?
        msg.getElementAsFloat64("EMSX_BROKER_COMM") : 0;
    double emsx_bse_avg_price =
        msg.hasElement("EMSX_BSE_AVG_PRICE") ?
        msg.getElementAsFloat64("EMSX_BSE_AVG_PRICE") : 0;
    int emsx_bse_filled = msg.hasElement("EMSX_BSE_FILLED") ?
        msg.getElementAsInt32("EMSX_BSE_FILLED") : 0;
    String emsx_clearing_account =
        msg.hasElement("EMSX_CLEARING_ACCOUNT") ?
        msg.getElementAsString("EMSX_CLEARING_ACCOUNT") : "";
    String emsx_clearing_firm =
        msg.hasElement("EMSX_CLEARING_FIRM") ?
        msg.getElementAsString("EMSX_CLEARING_FIRM") : "";
    String emsx_comm_diff_flag =
        msg.hasElement("EMSX_COMM_DIFF_FLAG") ?
        msg.getElementAsString("EMSX_COMM_DIFF_FLAG") : "";
    double emsx_comm_rate = msg.hasElement("EMSX_COMM_RATE") ?
        msg.getElementAsFloat64("EMSX_COMM_RATE") : 0;
    String emsx_currency_pair =
        msg.hasElement("EMSX_CURRENCY_PAIR") ?
        msg.getElementAsString("EMSX_CURRENCY_PAIR") : "";
    String emsx_custom_account =
        msg.hasElement("EMSX_CUSTOM_ACCOUNT") ?
        msg.getElementAsString("EMSX_CUSTOM_ACCOUNT") : "";
    double emsx_day_avg_price =
        msg.hasElement("EMSX_DAY_AVG_PRICE") ?
        msg.getElementAsFloat64("EMSX_DAY_AVG_PRICE") : 0;
    int emsx_day_fill = msg.hasElement("EMSX_DAY_FILL") ?
        msg.getElementAsInt32("EMSX_DAY_FILL") : 0;
    String emsx_exchange_destination =
        msg.hasElement("EMSX_EXCHANGE_DESTINATION") ?
        msg.getElementAsString("EMSX_EXCHANGE_DESTINATION") : "";
```



```
String emsx_exec_instruction =
    msg.hasElement("EMSX_EXEC_INSTRUCTION") ?
    msg.getElementAsString("EMSX_EXEC_INSTRUCTION") : "";
String emsx_execute_broker =
    msg.hasElement("EMSX_EXECUTE_BROKER") ?
    msg.getElementAsString("EMSX_EXECUTE_BROKER") : "";
int emsx_fill_id = msg.hasElement("EMSX_FILL_ID") ?
    msg.getElementAsInt32("EMSX_FILL_ID") : 0;
int emsx_filled = msg.hasElement("EMSX_FILLED") ?
    msg.getElementAsInt32("EMSX_FILLED") : 0;
int emsx_gtd_date = msg.hasElement("EMSX_GTD_DATE") ?
    msg.getElementAsInt32("EMSX_GTD_DATE") : 0;
String emsx_hand_instruction =
    msg.hasElement("EMSX_HAND_INSTRUCTION") ?
    msg.getElementAsString("EMSX_HAND_INSTRUCTION") : "";
int emsx_is_manual_route =
    msg.hasElement("EMSX_IS_MANUAL_ROUTE") ?
    msg.getElementAsInt32("EMSX_IS_MANUAL_ROUTE") : 0;
int emsx_last_fill_date =
    msg.hasElement("EMSX_LAST_FILL_DATE") ?
    msg.getElementAsInt32("EMSX_LAST_FILL_DATE") : 0;
int emsx_last_fill_time =
    msg.hasElement("EMSX_LAST_FILL_TIME") ?
    msg.getElementAsInt32("EMSX_LAST_FILL_TIME") : 0;
String emsx_last_market = msg.hasElement("EMSX_LAST_MARKET") ?
    msg.getElementAsString("EMSX_LAST_MARKET") : "";
double emsx_last_price = msg.hasElement("EMSX_LAST_PRICE") ?
    msg.getElementAsFloat64("EMSX_LAST_PRICE") : 0;
int emsx_last_shares = msg.hasElement("EMSX_LAST_SHARES") ?
    msg.getElementAsInt32("EMSX_LAST_SHARES") : 0;
double emsx_limit_price = msg.hasElement("EMSX_LIMIT_PRICE") ?
    msg.getElementAsFloat64("EMSX_LIMIT_PRICE") : 0;
double emsx_misc_fees = msg.hasElement("EMSX_MISC_FEES") ?
    msg.getElementAsFloat64("EMSX_MISC_FEES") : 0;
int emsx_ml_leg_quantity =
    msg.hasElement("EMSX_ML_LEG_QUANTITY") ?
    msg.getElementAsInt32("EMSX_ML_LEG_QUANTITY") : 0;
int emsx_ml_num_legs = msg.hasElement("EMSX_ML_NUM_LEGS") ?
    msg.getElementAsInt32("EMSX_ML_NUM_LEGS") : 0;
double emsx_ml_percent_filled =
    msg.hasElement("EMSX_ML_PERCENT_FILLED") ?
    msg.getElementAsFloat64("EMSX_ML_PERCENT_FILLED") : 0;
double emsx_ml_ratio = msg.hasElement("EMSX_ML_RATIO") ?
    msg.getElementAsFloat64("EMSX_ML_RATIO") : 0;
double emsx_ml_remain_balance =
    msg.hasElement("EMSX_ML_REMAIN_BALANCE") ?
    msg.getElementAsFloat64("EMSX_ML_REMAIN_BALANCE") : 0;
String emsx_ml_strategy = msg.hasElement("EMSX_ML_STRATEGY") ?
    msg.getElementAsString("EMSX_ML_STRATEGY") : "";
int emsx_ml_total_quantity =
    msg.hasElement("EMSX_ML_TOTAL_QUANTITY") ?
    msg.getElementAsInt32("EMSX_ML_TOTAL_QUANTITY") : 0;
String emsx_notes = msg.hasElement("EMSX_NOTES") ?
    msg.getElementAsString("EMSX_NOTES") : "";
double emsx_nse_avg_price =
    msg.hasElement("EMSX_NSE_AVG_PRICE") ?
    msg.getElementAsFloat64("EMSX_NSE_AVG_PRICE") : 0;
int emsx_nse_filled = msg.hasElement("EMSX_NSE_FILLED") ?
    msg.getElementAsInt32("EMSX_NSE_FILLED") : 0;
```

```
String emsx_order_type = msg.hasElement("EMSX_ORDER_TYPE") ?
    msg.getElementAsString("EMSX_ORDER_TYPE") : "";
String emsx_p_a = msg.hasElement("EMSX_P_A") ?
    msg.getElementAsString("EMSX_P_A") : "";
double emsx_percent_remain =
    msg.hasElement("EMSX_PERCENT_REMAIN") ?
    msg.getElementAsFloat64("EMSX_PERCENT_REMAIN") : 0;
double emsx_principal = msg.hasElement("EMSX_PRINCIPAL") ?
    msg.getElementAsFloat64("EMSX_PRINCIPAL") : 0;
int emsx_queued_date = msg.hasElement("EMSX_QUEUED_DATE") ?
    msg.getElementAsInt32("EMSX_QUEUED_DATE") : 0;
int emsx_queued_time = msg.hasElement("EMSX_QUEUED_TIME") ?
    msg.getElementAsInt32("EMSX_QUEUED_TIME") : 0;
String emsx_reason_code = msg.hasElement("EMSX_REASON_CODE") ?
    msg.getElementAsString("EMSX_REASON_CODE") : "";
String emsx_reason_desc = msg.hasElement("EMSX_REASON_DESC") ?
    msg.getElementAsString("EMSX_REASON_DESC") : "";
double emsx_remain_balance =
    msg.hasElement("EMSX_REMAIN_BALANCE") ?
    msg.getElementAsFloat64("EMSX_REMAIN_BALANCE") : 0;
int emsx_route_create_date =
    msg.hasElement("EMSX_ROUTE_CREATE_DATE") ?
    msg.getElementAsInt32("EMSX_ROUTE_CREATE_DATE") : 0;
int emsx_route_create_time =
    msg.hasElement("EMSX_ROUTE_CREATE_TIME") ?
    msg.getElementAsInt32("EMSX_ROUTE_CREATE_TIME") : 0;
int emsx_route_id = msg.hasElement("EMSX_ROUTE_ID") ?
    msg.getElementAsInt32("EMSX_ROUTE_ID") : 0;
int emsx_route_last_update_time =
    msg.hasElement("EMSX_ROUTE_LAST_UPDATE_TIME") ?
    msg.getElementAsInt32("EMSX_ROUTE_LAST_UPDATE_TIME") : 0;
double emsx_route_price = msg.hasElement("EMSX_ROUTE_PRICE") ?
    msg.getElementAsFloat64("EMSX_ROUTE_PRICE") : 0;
int emsx_sequence = msg.hasElement("EMSX_SEQUENCE") ?
    msg.getElementAsInt32("EMSX_SEQUENCE") : 0;
double emsx_settle_amount =
    msg.hasElement("EMSX_SETTLE_AMOUNT") ?
    msg.getElementAsFloat64("EMSX_SETTLE_AMOUNT") : 0;
int emsx_settle_date = msg.hasElement("EMSX_SETTLE_DATE") ?
    msg.getElementAsInt32("EMSX_SETTLE_DATE") : 0;
String emsx_status = msg.hasElement("EMSX_STATUS") ?
    msg.getElementAsString("EMSX_STATUS") : "";
double emsx_stop_price = msg.hasElement("EMSX_STOP_PRICE") ?
    msg.getElementAsFloat64("EMSX_STOP_PRICE") : 0;
int emsx_strategy_end_time =
    msg.hasElement("EMSX_STRATEGY_END_TIME") ?
    msg.getElementAsInt32("EMSX_STRATEGY_END_TIME") : 0;
double emsx_strategy_part_rate1 =
    msg.hasElement("EMSX_STRATEGY_PART_RATE1") ?
    msg.getElementAsFloat64("EMSX_STRATEGY_PART_RATE1") : 0;
double emsx_strategy_part_rate2 =
    msg.hasElement("EMSX_STRATEGY_PART_RATE2") ?
    msg.getElementAsFloat64("EMSX_STRATEGY_PART_RATE2") : 0;
int emsx_strategy_start_time =
    msg.hasElement("EMSX_STRATEGY_START_TIME") ?
    msg.getElementAsInt32("EMSX_STRATEGY_START_TIME") : 0;
String emsx_strategy_style =
    msg.hasElement("EMSX_STRATEGY_STYLE") ?
    msg.getElementAsString("EMSX_STRATEGY_STYLE") : "";
```

```

String emsx_strategy_type =
    msg.hasElement("EMSX_STRATEGY_TYPE") ?
    msg.getElementAsString("EMSX_STRATEGY_TYPE") : "";
String emsx_tif = msg.hasElement("EMSX_TIF") ?
    msg.getElementAsString("EMSX_TIF") : "";
int emsx_time_stamp = msg.hasElement("EMSX_TIME_STAMP") ?
    msg.getElementAsInt32("EMSX_TIME_STAMP") : 0;
String emsx_type = msg.hasElement("EMSX_TYPE") ?
    msg.getElementAsString("EMSX_TYPE") : "";
int emsx_urgency_level = msg.hasElement("EMSX_URGENCY_LEVEL") ?
    msg.getElementAsInt32("EMSX_URGENCY_LEVEL") : 0;
double emsx_user_comm_amount =
    msg.hasElement("EMSX_USER_COMM_AMOUNT") ?
    msg.getElementAsFloat64("EMSX_USER_COMM_AMOUNT") : 0;
double emsx_user_comm_rate =
    msg.hasElement("EMSX_USER_COMM_RATE") ?
    msg.getElementAsFloat64("EMSX_USER_COMM_RATE") : 0;
double emsx_user_fees = msg.hasElement("EMSX_USER_FEES") ?
    msg.getElementAsFloat64("EMSX_USER_FEES") : 0;
double emsx_user_net_money =
    msg.hasElement("EMSX_USER_NET_MONEY") ?
    msg.getElementAsFloat64("EMSX_USER_NET_MONEY") : 0;
int emsx_working = msg.hasElement("EMSX_WORKING") ?
    msg.getElementAsInt32("EMSX_WORKING") : 0;

System.out.println("ROUTE MESSAGE: CorrelationID(" +
    msg.correlationID() + ") EMSX_SEQUENCE: " +
    Integer.toString(emsx_sequence) + "\tROUTE ID: " +
    Integer.toString(emsx_route_id) + "\t EVENT_STATUS: " +
    event_status + "\tEMSX_STATUS: " + emsx_status);

    }
}
else
{
    System.err.println("Error: Unexpected message");
}
}

private void processResponseEvent(Event evt, Session session)
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {
        Message msg = msgIter.next();

        if (msg.correlationID() == refdata_req_id)
        {
            System.out.println("RefData RESPONSE received...");

            Element securities = msg.getElement(SEcurity_DATA);
            Element security = securities.getValueAsElement(0);
            Element fields = security.getElement(FIELD_DATA);
            String ticker_desc =

```

```

        fields.getElementAsString("PARSEKYABLE_DES_SOURCE");

        System.out.println("Ticker Description: " + ticker_desc);
    }
    else
    {

        CorrelationID matchCorID = msg.correlationID();

        if (msg.messageType().equals(ERROR_INFO))
        {
            int errorCode = msg.getElementAsInt32("ERROR_CODE");
            String errorMessage = msg.getElementAsString("ERROR_MESSAGE");

            System.out.println("RESPONSE ERROR\nERROR CODE: " + errorCode +
                "\tERROR MESSAGE: " + errorMessage);
        }
        else if (msg.messageType().equals(CREATE_ORDER_AND_ROUTE))
        {
            System.out.println("CREATE_ORDER_AND_ROUTE Response received...");

            int emsx_sequence = msg.getElementAsInt32("EMSX_SEQUENCE");
            int emsx_route_id = msg.getElementAsInt32("EMSX_ROUTE_ID");
            String message = msg.getElementAsString("MESSAGE");

            System.out.println("EMSX_SEQUENCE: " +
                Integer.toString(emsx_sequence) + "\tEMSX_ROUTE_ID: " +
                Integer.toString(emsx_route_id) + "\tMESSAGE: " + message);

            IDSequenceMap matched = getMapEntry(matchCorID);

            if (!matched.equals(null))
            {
                System.out.println("Request ID recognised(" + matched.requestID
                    + ")...assigning SEQUENCE and ROUTE ID");
                matched.sequenceNo = emsx_sequence;
                matched.routeID = emsx_route_id;
            }
            else
            {
                System.out.println("Request ID not recognised(" + matchCorID +
                    ")...ignoring");
            }
        }
    }
}

private IDSequenceMap getMapEntry(CorrelationID id)
{
    Iterator<IDSequenceMap> itr = idMapping.iterator();

    while(itr.hasNext())
    {
        IDSequenceMap find = itr.next();
        if (find.requestID.equals(id))
        {

```

Bloomberg

```
        return find;
    }
}

return null;
}

private void processMiscEvent(Event evt, Session session)
{
    System.out.println("Processing " + evt.eventType().toString());

    MessageIterator msgIter = evt.messageIterator();
    while (msgIter.hasNext()) {

        Message msg = msgIter.next();

        System.out.println("MESSAGE: " + msg);
    }
}

private boolean parseCommandLine(String[] args)
{
    if (args.length < 4)
    {
        System.out.println("Error: Missing required parameters\n");
        printUsage();
        return false;
    }
    else
    {
        for (int i = 0; i < args.length; i++)
        {
            if (isArg(args[i], "SELLAT")) sellat =
                Double.parseDouble(getArgValue(args[i]));
            else if (isArg(args[i], "BUYAT")) buyat =
                Double.parseDouble(getArgValue(args[i]));
            else if (isArg(args[i], "AMOUNT")) amount =
                Integer.parseInt(getArgValue(args[i]));
            else if (isArg(args[i], "TICKER")) ticker = getArgValue(args[i]);
            else System.out.println("Warning>> Unknown parameter:" + args[i]);
        }

        showParameters();
    }
    return true;
}

private void showParameters()
{
    System.out.println("Parameter List:-");
    System.out.println("SELLAT: " + sellat);
    System.out.println("BUYAT: " + buyat);
    System.out.println("AMOUNT: " + amount);
    System.out.println("TICKER: " + ticker);
}

private boolean isArg(String arg, String find)
```

Bloomberg

```
{
    if (arg.indexOf(find) >= 0) return true;
    else return false;
}

private String getArgValue(String arg)
{
    return (arg.substring(arg.indexOf("=") + 1));
}

private void printUsage()
{
    System.out.println("Usage:");
    System.out.println("EMSXCreateOrderAsyncRequest SELLAT=<value> BUYAT=<value>
    AMOUNT=<value> TICKER=\"<value>\"");
}
}
```

Bloomberg

Python & EMSX API

EMSX API is also accessible from Python. Go to the following link and install the Python binding in your desktop.

<http://www.bloomberglabs.com/api/>

Currently Bloomberg only supports Python 2.6 & 2.7.

How to setup the Python BLPAPI

These instructions outline how to install the Bloomberg BLPAPI for Python on Windows machine with no C++ compiler. This should work for both desktop & server API in both 32 and 64 bit.

DOWNLOAD & INSTALL

- 1) Install Python v2 itself.

Note that Python v3 will not work with the BLPAPI SDK and the setup.py explicitly checks the version.

- Go to <https://www.python.org/downloads/windows/>
 - Find the 32 bit/64 bit version under "Windows x86 MSI installer", which points to <https://www.python.org/ftp/python/2.7.9/python-2.7.9.msi>
 - Download and install that msi taking all defaults.
 - Open a *new* command window.
Start→Run→cmd
 - Create and run a python test program as follows:
C:\> echo print "hello world" > test.py
run the program, verify the output
C:\> test.py
hello world
- 2) If you do not have software to extract an ISO file then install 7-zip from the software catalog (Bloomberg internal) or from <http://www.7-zip.org/>
 - 3) Install Visual Studio 2010 C++ Express (or Pro, etc). Do not use a higher version. (e.g. 2013 as it is not currently supported)
 - <http://go.microsoft.com/?linkid=9709969>
 - This gives the file **VS2010Express1.iso**
 - Right click and "extract to VS2010Express1" using 7z, this creates a folder with the extracted iso.
 - Browse the extracted folder and run Setup.hta, selecting C++.

Bloomberg

4) Download and install the Microsoft Visual Studio 2010 Service Pack 1

- If you have issues with 2010 C++, please install the following.
- <http://go.microsoft.com/fwlink/?LinkId=210710>
- This provides the file VS2010SP1dvd1.iso; please note that this is a 1.5GB file.
- Right click the downloaded file and "extract to VS2010SP1dvd1" using 7zip, this creates a folder with the extracted iso.
- Browse the extracted folder and run Setup.exe.
- Go to WAPI<go> on the Bloomberg Terminal and download the BPIPE SDK, which comes as a zip file. Extract the zip file into a folder, e.g. C:\NoAdmin_ServerAPI_SDK

TEST

Please use **Wordpad** to open the README file found in the following folder

- C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5
(Please DO NOT use Notepad as the file is Unix 'nl' line endings rather than the usual Windows 'crlf' line endings, which forces Notepad to wrap all lines regardless of settings.)
- Follow the instructions carefully.

1) Setting the BLPAPI_ROOT variable:

- Start→Control Panel → System → Advanced System Settings →Environment Variables
→User variables
- Click New to create a new variable and enter:
Variable name: BLPAPI_ROOT
Variable value: C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\C++API\v3.7 .5.1
- Start a *new* command line session.
Start→Run→cmd
- check that the environment variable is set by entering "set b" without the quotes:
C:\> set b

You should see the following response:

```
BLPAPI_ROOT= C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\C++API\v3.7 .5.1
```

2) Check the files in the python directory

- C:\> cd /d C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5
- C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5>dir
Volume in drive C is Windows
Volume Serial Number is D23C-720D

```
Directory of C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5
```

```
07/01/2015 11:06 <DIR> .
```


Bloomberg

```
07/01/2015 11:06 <DIR>    ..
07/01/2015 11:06 <DIR>    blpapi
07/02/2014 15:38        514 changelog.txt
07/01/2015 11:06 <DIR>    examples
26/09/2012 09:01        6,834 NOTES
07/02/2014 15:42        503 PKG-INFO
29/01/2014 16:11        5,469 README
26/09/2012 09:01        61 setup.cfg
07/02/2014 15:38        2,317 setup.py
```

3) Run the setup.py program

- Now run the script. You should see output similar to the following:

```
C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5>setup.py
install
running install
running build
running build_py
creating build
creating build\lib.win32-2.7
creating build\lib.win32-2.7\blpapi
copying blpapi\abstractsession.py -> build\lib.win32-2.7\blpapi
copying blpapi\constant.py -> build\lib.win32-2.7\blpapi
copying blpapi\datatype.py -> build\lib.win32-2.7\blpapi
copying blpapi\datetime.py -> build\lib.win32-2.7\blpapi
copying blpapi\element.py -> build\lib.win32-2.7\blpapi
copying blpapi\event.py -> build\lib.win32-2.7\blpapi
copying blpapi\eventdispatcher.py -> build\lib.win32-2.7\blpapi
copying blpapi\eventformatter.py -> build\lib.win32-2.7\blpapi
copying blpapi\exception.py -> build\lib.win32-2.7\blpapi
copying blpapi\identity.py -> build\lib.win32-2.7\blpapi
copying blpapi\internals.py -> build\lib.win32-2.7\blpapi
copying blpapi\message.py -> build\lib.win32-2.7\blpapi
copying blpapi\name.py -> build\lib.win32-2.7\blpapi
copying blpapi\providersession.py -> build\lib.win32-2.7\blpapi
copying blpapi\request.py -> build\lib.win32-2.7\blpapi
copying blpapi\resolutionlist.py -> build\lib.win32-2.7\blpapi
copying blpapi\schema.py -> build\lib.win32-2.7\blpapi
copying blpapi\service.py -> build\lib.win32-2.7\blpapi
copying blpapi\session.py -> build\lib.win32-2.7\blpapi
copying blpapi\sessionoptions.py -> build\lib.win32-2.7\blpapi
copying blpapi\subscriptionlist.py -> build\lib.win32-2.7\blpapi
copying blpapi\topic.py -> build\lib.win32-2.7\blpapi
copying blpapi\topiclist.py -> build\lib.win32-2.7\blpapi
copying blpapi\utils.py -> build\lib.win32-2.7\blpapi
```

Bloomberg

```
copying blpapi\__init__.py -> build\lib.win32-2.7\blpapi
running build_ext
building 'blpapi._internals' extension
creating build\temp.win32-2.7
creating build\temp.win32-2.7\Release
creating build\temp.win32-2.7\Release\blpapi
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3
/GS- /DNDEBUG -IC:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\C++API\v3.7.5.1\include -
IC:\Python27\include -IC:\Python27\PC\Tpb\blpapi\internals_wrap.cxx
/Fobuild\temp.win32-2.7\Release\blpapi\internals_wrap.obj
internals_wrap.cxx
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\INCLUDE\xlocale(323) : warning
C4530: C++ exception handler used, but unwind semantics are not enabled. Specify /EHsc
blpapi\internals_wrap.cxx(3907) : warning C4244: 'argument' : conversion from
'blpapi_Float64_t' to 'blpapi_Float32_t', possible loss of data
blpapi\internals_wrap.cxx(3921) : warning C4244: 'argument' : conversion from
'blpapi_Float64_t' to 'blpapi_Float32_t', possible loss of data
blpapi\internals_wrap.cxx(3940) : warning C4244: 'argument' : conversion from
'blpapi_Float64_t' to 'blpapi_Float32_t', possible loss of data
blpapi\internals_wrap.cxx(3994) : warning C4244: 'argument' : conversion from
'blpapi_Float64_t' to 'blpapi_Float32_t', possible loss of data
blpapi\internals_wrap.cxx(4011) : warning C4244: 'argument' : conversion from
'blpapi_Float64_t' to 'blpapi_Float32_t', possible loss of data
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\BIN\link.exe /DLL /nologo
/INCREMENTAL:NO
/LIBPATH:C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\C++API\v3.7.5.1\lib
/LIBPATH:C:\Python27\libs /LIBPATH:C:\Python27\PCbuild blpapi3_32.lib
/EXPORT:init_internals build\temp.win32-2.7\Release\blpapi\internals_wrap.obj
/OUT:build\lib.win32-2.7\blpapi\_internals.pyd /IMPLIB:build\temp.win32-
2.7\Release\blpapi\_internals.lib /MANIFESTFILE:build\temp.win32-
2.7\Release\blpapi\_internals.pyd.manifest /MANIFEST
Creating library build\temp.win32-2.7\Release\blpapi\_internals.lib and object
build\temp.win32-2.7\Release\blpapi\_internals.exp
running install_lib
creating C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\abstractsession.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\constant.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\datatype.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\datetime.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\element.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\event.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\eventdispatcher.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\eventformatter.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\exception.py -> C:\Python27\Lib\site-packages\blpapi
```

Bloomberg

```
copying build\lib.win32-2.7\blpapi\identity.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\internals.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\message.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\name.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\providersession.py ->
C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\request.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\resolutionlist.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\schema.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\service.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\session.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\sessionoptions.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\subscriptionlist.py -> C:\Python27\Lib\site-
packages\blpapi
copying build\lib.win32-2.7\blpapi\topic.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\topiclist.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\utils.py -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\_internals.pyd -> C:\Python27\Lib\site-packages\blpapi
copying build\lib.win32-2.7\blpapi\__init__.py -> C:\Python27\Lib\site-packages\blpapi
byte-compiling C:\Python27\Lib\site-packages\blpapi\abstractsession.py to
abstractsession.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\constant.py to constant.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\datatype.py to datatype.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\datetime.py to datetime.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\element.py to element.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\event.py to event.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\eventdispatcher.py to
eventdispatcher.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\eventformatter.py to
eventformatter.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\exception.py to exception.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\identity.py to identity.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\internals.py to internals.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\message.py to message.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\name.py to name.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\providersession.py to
providersession.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\request.py to request.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\resolutionlist.py to resolutionlist.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\schema.py to schema.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\service.py to service.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\session.py to session.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\sessionoptions.py to
sessionoptions.pyc
```

Bloomberg

```
byte-compiling C:\Python27\Lib\site-packages\blpapi\subscriptionlist.py to
subscriptionlist.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\topic.py to topic.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\topiclist.py to topiclist.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\utils.py to utils.pyc
byte-compiling C:\Python27\Lib\site-packages\blpapi\__init__.py to __init__.pyc
running install_egg_info
Writing C:\Python27\Lib\site-packages\blpapi-3.5.5-py2.7.egg-info

C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5>
```

4) Run a sample:

- C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5> cd examples

```
C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5\examples>
RefDataTableOverrideExample.py
RefDataTableOverrideExample
Connecting to localhost:8194
Sending Request: ReferenceDataRequest = {
  securities[] = {
    "CWHL 2006-20 1A1 Mtge"
  }
  fields[] = {
    "MTG_CASH_FLOW", "SETTLE_DT"
  }
  overrides[] = {
    overrides = {
      fieldId = "ALLOW_DYNAMIC_CASHFLOW_CALC"
      value = "Y"
    }
    overrides = {
      fieldId = "LOSS_SEVERITY"
      value = "31"
    }
  }
  tableOverrides[] = {
    tableOverrides = {
      fieldId = "DEFAULT_VECTOR"
      row[] = {
        row = {
          value[] = {
            "Anchor", "PROJ"
          }
        }
      }
      row = {
```

Bloomberg

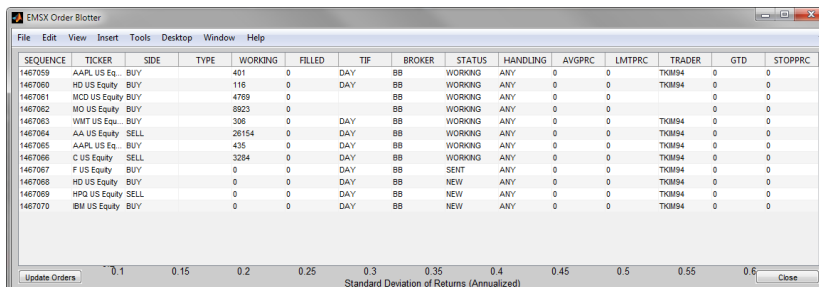
```
        value[] = {
            "Type", "CDR"
        }
    }
    row = {
        value[] = {
            "1.000000", "12", "S"
        }
    }
    row = {
        value[] = {
            "2.000000", "12", "R"
        }
    }
}
}
```

CWHL 2006-20 1A1 Mtge

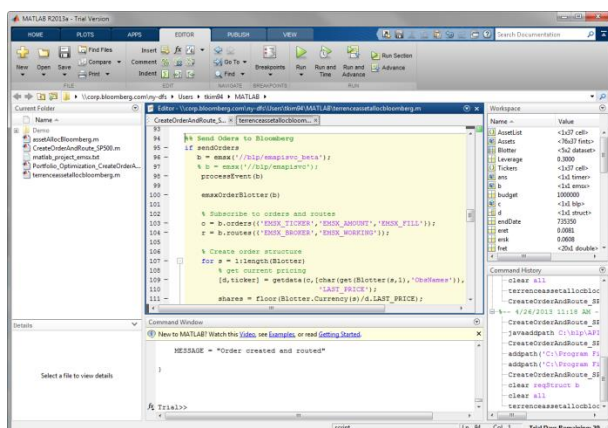
C:\NoAdmin_ServerAPI_SDK\ServerAPI\APIv3\Python\v3.5.5\examples>

MATLAB & EMSX API

As of March 7, 2013, MathWorks will be releasing the Trading Toolbox as part of the MATLAB Q1 2013 release.

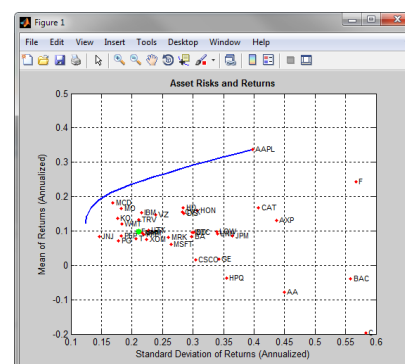


SEQUENCE	TICKER	SIDE	TYPE	WORKING	FILLED	TIF	BROKER	STATUS	HANDLING	AVGPRC	LMTPRC	TRADER	GTD	STOPPRC
1467059	AAPL US Eq.	BUY		401	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467060	HD US Equity	BUY		116	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467061	MCD US Equity	BUY		4789	0		BB	WORKING	ANY	0	0	0	0	0
1467062	MO US Equity	BUY		8923	0		BB	WORKING	ANY	0	0	0	0	0
1467063	WMT US Eq.	BUY		308	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467064	AA US Equity	SELL		26154	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467065	AAPL US Eq.	BUY		435	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467066	C US Equity	SELL		3284	0	DAY	BB	WORKING	ANY	0	0	TKM94	0	0
1467067	F US Equity	BUY		0	0	DAY	BB	SENT	ANY	0	0	TKM94	0	0
1467068	HD US Equity	BUY		0	0	DAY	BB	NEW	ANY	0	0	TKM94	0	0
1467069	HPQ US Equity	SELL		0	0	DAY	BB	NEW	ANY	0	0	TKM94	0	0
1467070	IBM US Equity	BUY		0	0	DAY	BB	NEW	ANY	0	0	TKM94	0	0



This toolbox will allow MATLAB users to access EMSX API directly from MATLAB to manage and automate equities, futures and options trades.

The users can quickly pull in real-time and historical data from the Bloomberg Professional Service, build and test strategies using the data, and then execute strategies using the EMSX API.



Fills over EMSX API (//blp/emsx.history)

The emsx.history service call will provide individual fill information via request / response service. Please reference emsx.history.html for the schema details along with emsx.history code samples.

For emsx.history services to work the user needs to have an EMSI<GO> Fills Export profile before his fills starts to record in the EMSX database.



Available Fields:

Account	Trading account used in EMSX
BasketName	Basket Name set in EMSX
CorrectedFillId	Corrected fill ID
DateTimeOfFill	Date and time of fill
Exchange	Exchange where it was traded
Executing Broker	Executing broker name
FillId	Fill ID
FillPrice	Fill Price
IsLeg	For multi-leg orders (futures & options)
LastCapacity	Last capacity
LastMarket	Last Market / Executed Venue
Liquidity	Liquidity code indicator (maker vs. taker) 1 = Added Liquidity

Bloomberg

	2 = Removed Liquidity 3 = Liquidity Routed Out 4 = Auction
MultilegId	ID of the multi-leg order
OrderID	Order ID (EMSX_SEQUENCE)
ReroutedBroker	Name of the re-routed broker
RouteID	Placement ID
Side	Order Side (e.g. Buy, Sell, Short Sell, Buy to Cover, and etc.)
Ticker	Symbol of the security
TIF	Time in force (e.g. day, GTC, GTD and etc.)
TraderUuid	UUID of the trader
Type	Order Type

Failover

The Bloomberg data center provides a Hot-Hot backup to the API subscription services. The BBCOMM executable on the user's desktop subscribes to a server maintaining API services at Bloomberg's data center. Bloomberg's API service then notifies the user's BBCOMM as to which API service is designated as the primary and where to redirect in the case where primary API services are unavailable.

This is done behind the scene by the user's desktop BBCOMM executable. This way the user application does not need to concern itself with re-establishing a session in the event that the primary API service is unavailable. If the primary goes down, all API traffic is automatically redirected to Bloomberg's backup API service.

If there is a break in service on the primary connection, then the API infrastructure immediately sends subscription request(s) on behalf of all the subscriptions that were subscribed to the primary server. After the backup session is active and considered the new primary, the Bloomberg API service will send Initial Paint messages to client application. The client application is not notified about the change in primary to backup service. Thus, client application should be ready to process the Initial Paint messages at any time. Heartbeat messages are sent to the client every 30 seconds and can be used to identify a problem with the real time service.

Bloomberg

Time Zone in EMSX API

All time fields in EMSX API are set to the user time zone. This can be set in there Bloomberg terminal from **TZDF<GO>**.

Extended Requests

In the request / responses, you can see the “...Ex” requests in the schema of *emapisvc* and *emapisvc_beta*. These are extension of the existing requests. To make the EMSX API backward compatible, some of the newer features are added under new requests as an extension. Following are the extended requests: CreateOrderAndRouteEx, GroupRouteEx, ModifyRouteEx, and RouteEx.

FAQ

1. 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.

2. 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.

3. 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>**

4. 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)

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

10. 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 field in the first place or is requesting for a *static* field.

11. 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 an order or route within their application.

12. How do I receive Fill messages?

Currently, Fill level messaging is not supported. A fill event will generate a **UPD_ORDER_ROUTE** message, with the applicable changes to the order and route data. The only way to track individual fills is to monitor these messages and record changes as they occur. This way, the application can calculate the changes from one fill to the next to identify and track the individual fills.

EMSX_LAST_XXXX fields are used for this purpose.

13. 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.

14. 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, they 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.

15. 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.

16. 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.

17. Why do I get “User ABCDE is not permitted for the API”?

EMSX Help Desk must enable users for EMSX API access via EMSS.

18. 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.

19. I don't see the EMSX button (Excel Add-In) in my Excel?

This is mostly due to the user not being enabled for EMSX API. Click Help Help on **EMSX<GO>** and ask the EMSX Help Desk personnel to see if your UUID is enabled for EMSX API Excel Add-In. If the user has multiple Excel Add-Ins, the EMSX button will be under the Trading Icon.



20. 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.

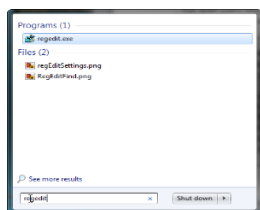
21. How do I restart BBCOMM?

- Close all instances of Excel, Word and PowerPoint.
- Open task manager and kill bxlai.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.

22. How do I regenerate apiregistry.ini file?

Open regedit from RUN window and Clear the "APIRegistryCRC32" registry value located at "HKEY_LOCAL_MACHINE\SOFTWARE\Bloomberg L.P.\Office Tools\Settings" or "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Bloomberg L.P.\Office Tools\Settings" on Windows 7.

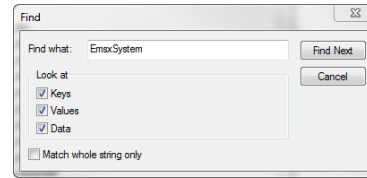
23. I am a Bloomberg AIM user and I am not able to connect from the Excel Add-In.



This is mostly often due to the AIM user not being able to connect to the beta environment (Y087). For AIM, users they will need to test in production since there are no AIM instance in the beta environment (Y087).

One of the ways to solve this is by going into the registry edit by clicking Start and type “regedit”.

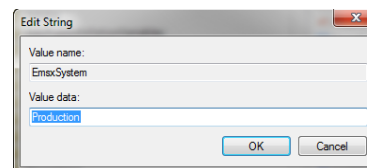
Once in the regedit.exe, click Alt-F and type “EmsxSystem”.



Double Click EmsxSystem

Name	Type	Data
(Default)	REG_SZ	6.1.00.28
CurrentLog	REG_SZ	
EmsxSystem	REG_SZ	Production
IsAPIUser	REG_SZ	Yes
IsTSPUser	REG_SZ	Yes
IsEtfFlag	REG_DWORD	0x00000000 (0)
IsHmfFlag	REG_DWORD	0x00000000 (0)
IsSpectrum	REG_DWORD	0xffffffff (4294967295)
SuspendRealTI...	REG_SZ	0
Tradebook	REG_DWORD	0x00000000 (0)
TradeSystem	REG_SZ	Production

Type the word Production in the Value Data column and Click OK.



24. 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.

25. How do I modify or reset the stop price of an order?

Set EMSX_STOP_PRICE to "-1" or -1

26. How do I reset my order from Limit to Market?

EMSX_LIMIT_PRICE=-99999 is only required when modifying *from* LMT to something else.

27. How...EMSX_RELEASE_TIME

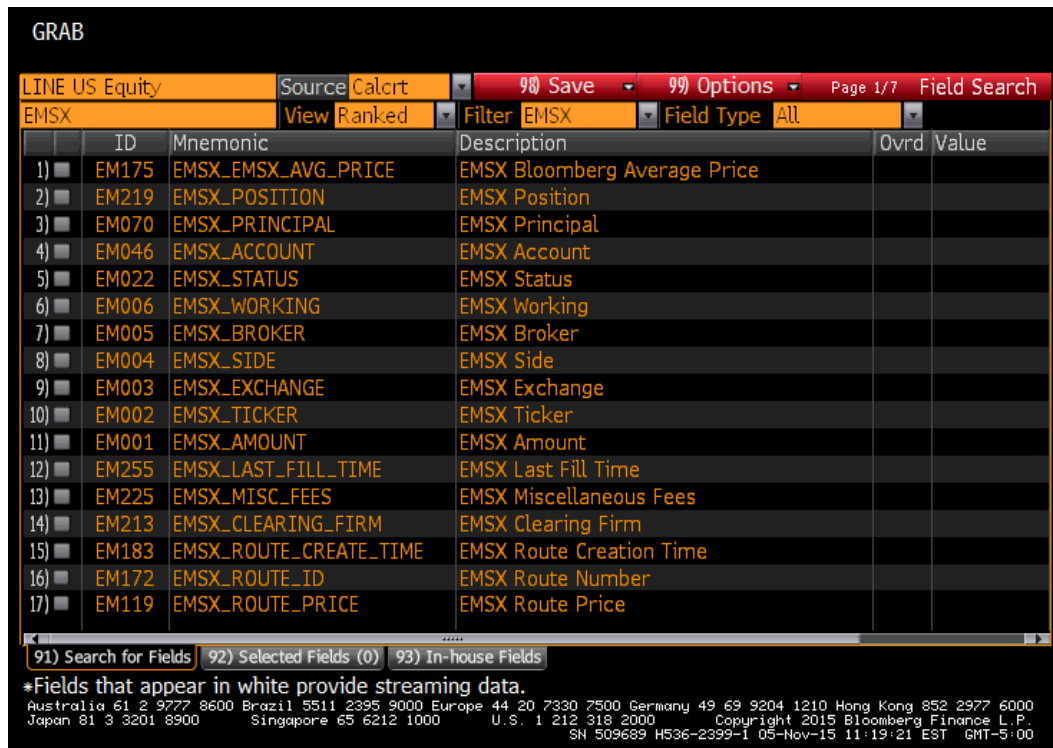
EMSX_RELEASE_TIME is in HH:MM format. For the API it is defaulted to the exchange time.

Bloomberg

Accessing Field Meta Data

For information about EMSX, fields please type **FLDS<GO>** inside your Bloomberg terminal.

Once in the **FLDS<GO>**, type **EMSX** underneath the security and choose **EMSX** under **Filter**.



GRAB

LINE	US Equity	Source	Calcut	90 Save	99 Options	Page 1/7	Field Search
EMSX		View	Ranked	Filter	EMSX	Field Type	All
	ID	Mnemonic	Description	Ovrd	Value		
1)	EM175	EMSX_EMSX_AVG_PRICE	EMSX Bloomberg Average Price				
2)	EM219	EMSX_POSITION	EMSX Position				
3)	EM070	EMSX_PRINCIPAL	EMSX Principal				
4)	EM046	EMSX_ACCOUNT	EMSX Account				
5)	EM022	EMSX_STATUS	EMSX Status				
6)	EM006	EMSX_WORKING	EMSX Working				
7)	EM005	EMSX_BROKER	EMSX Broker				
8)	EM004	EMSX_SIDE	EMSX Side				
9)	EM003	EMSX_EXCHANGE	EMSX Exchange				
10)	EM002	EMSX_TICKER	EMSX Ticker				
11)	EM001	EMSX_AMOUNT	EMSX Amount				
12)	EM255	EMSX_LAST_FILL_TIME	EMSX Last Fill Time				
13)	EM225	EMSX_MISC_FEES	EMSX Miscellaneous Fees				
14)	EM213	EMSX_CLEARING_FIRM	EMSX Clearing Firm				
15)	EM183	EMSX_ROUTE_CREATE_TIME	EMSX Route Creation Time				
16)	EM172	EMSX_ROUTE_ID	EMSX Route Number				
17)	EM119	EMSX_ROUTE_PRICE	EMSX Route Price				

91) Search for Fields 92) Selected Fields (0) 93) In-house Fields

*Fields that appear in white provide streaming data.

Australia 61 2 9777 8600 Brazil 5511 2395 9000 Europe 44 20 7330 7500 Germany 49 69 9204 1210 Hong Kong 852 2977 6000
Japan 81 3 3201 8900 Singapore 65 6212 1000 U.S. 1 212 318 2000 Copyright 2015 Bloomberg Finance L.P.
SN 509689 H536-2399-1 05-Nov-15 11:19:21 EST GMT-5:00

The **Field Type** can be filtered by EMSX: ALL, EMSX: ROUTE, EMSX: ORDER and EMSX: FILL. In addition, the EMSX API provides programmatic access to information about all of the EMSX fields via the [GetAllFieldMetaData](#) request.

Bloomberg

EXCEL

Bloomberg

EMSX API – Excel Add-In

The EMSX API Excel Add-in allows users to enter multiple equity, futures and options securities into an Excel spreadsheet and execute trades based on various custom built strategies.

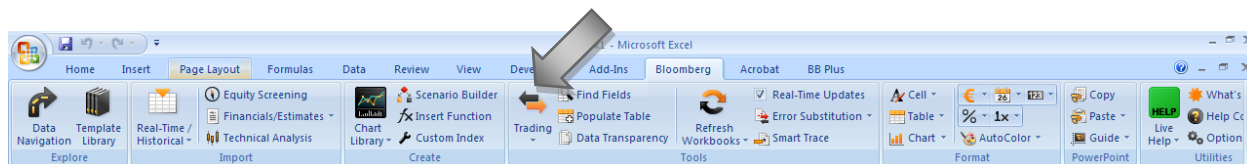
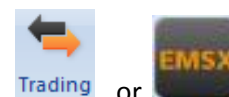
This is accomplished by creating a Staging Blotter using the Excel Add-In. The Staging Blotter is the entrance point where user can either manually or use offset method in VBA to enter the order information required to send to the various execution destination available in **EOR<GO>**.

The Order and Route Blotters are optional and they are designed to provide real-time update to the Orders and Routes similar to **EMSX<GO>**.

A user can setup multiple instances of the Excel Add-In as long as each instance creates in its own session of Excel. Excel Add-In supports Excel 2003 and later version of Microsoft Excel.

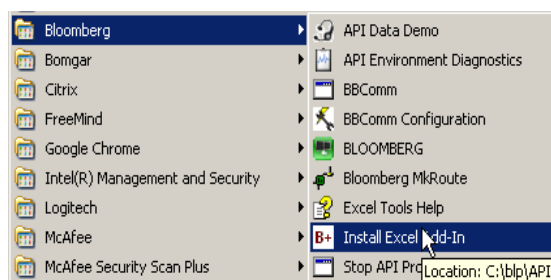
Getting Started with Excel Add-In

Once a user is enabled for EMSX API Excel Add-In, Inside the Bloomberg Ribbon there will be either Trading (If user has multiple applications installed) or Black EMSX Control Panel Button. Once clicked, the button will bring up the Control Panel for Excel Add-In.



If the user doesn't have the Bloomberg Ribbon in the Excel this can be manually installed by clicking
→ Start
→ All Programs
→ Bloomberg
→ Install Excel Add-In in Windows XP.

(Please Note Windows 7 or later will have slightly different menu navigation)



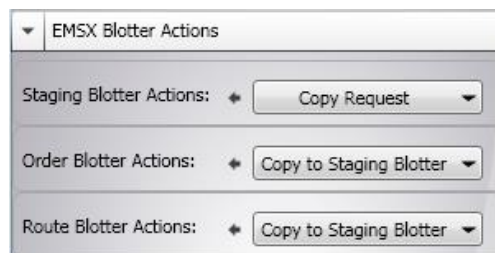
Once the EMSX API Excel Add-In is enabled for a particular UUID the Trading or EMSX button will show up as soon as the user has the Bloomberg Ribbon installed in Excel.

Bloomberg

EMSX Blotter Actions – Excel Add-In

EMSX Blotter Actions allow users to use the Control Panel to control and interact with Staging, Order and Route blotter.

Each Blotter Action will have a different menu under the drop down menu inside the control panel. These are the same actions as those the users would see when they right click on each work sheet.



EMSX Staging, Order and Route Blotters – Excel Add-In

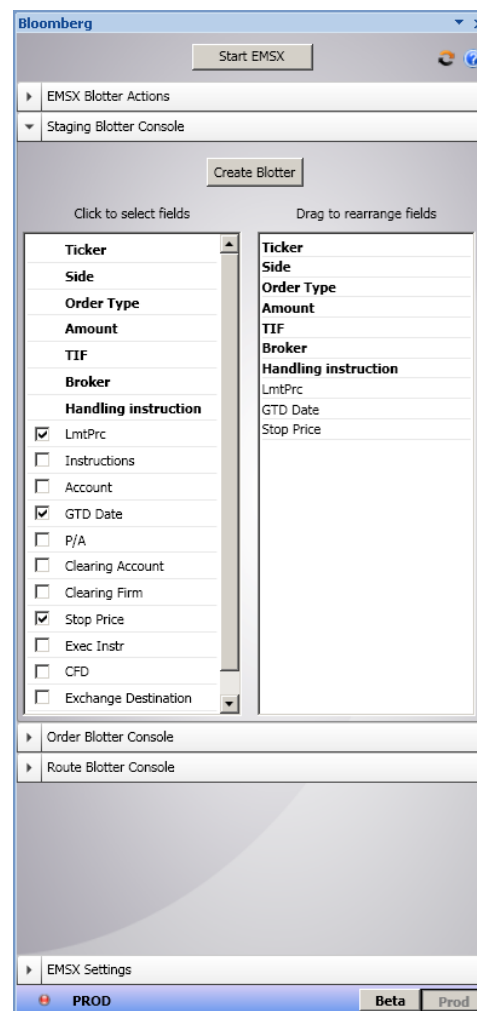
Using the Staging Blotter Console, Order Blotter Console, and Route Blotter Console the user can choose the fields they would like to expose to each worksheet.

The user cannot create Staging, Order and Route blotters inside the same worksheet. Each needs to be created in its own instance of the worksheet.

BOLD Faced fields are default fields you must have in your blotter and any other fields can be selected by checking the check box in each blotter console.

You can also run multiple instance of the Excel Add-In, one for each separate instance of Microsoft Excel running on your desktop.

The EMSX Excel Add-In will work for as long as at least one blotter exists within the workbook.
(Usually the Staging Blotter)



Bloomberg

EMSX Settings – Excel Add-In

Enable Team-View: The user can check Enable Team View and specify the Team Name the user belongs to. This will allow the user to view the Order and Route details for the Team. However, the user cannot trade on behalf of the team member in Excel Add-In.

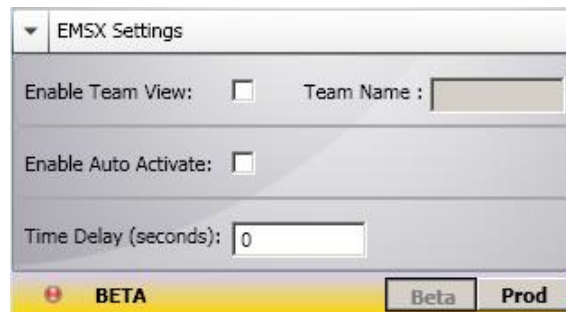
The Enable Team View in Excel Add-In is strictly to view orders and routes (placements) within the team the user belongs to. The user can still trade and modify his own orders and routes (placements) while the Team View is activated.

Auto-Activate: By checking Enable Auto Activate, the user eliminates the right click option to activate single or multiple roles inside the staging blotter.

Timed Delay: This will wait x number of seconds before activating.

Menu Bar: The Production Menu Bar will have the background color as Blue and the user can easily switch from Beta to Prod by clicking on the button.

Once the EMSX Engine is started, the user will not be able to toggle through the environment. The EMSX Engine button must be off in order for the user to switch from one environment to another.



Broker Strategies / Algorithms – Excel Add-In

Broker Strategies will be available in the Excel Add-In as of May 03, 2012 – Build 6517 and later. User can check the version of the build by clicking Live Help and About under the green HELP button inside the Bloomberg Ribbon in Excel. This will provide a splash screen with the date and build number.



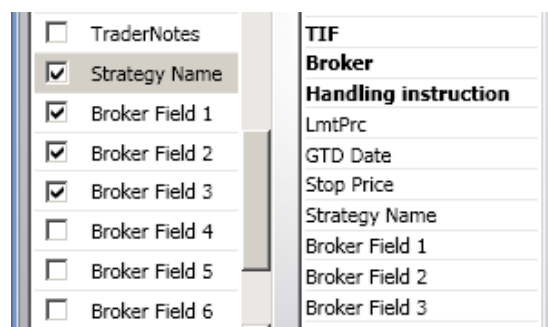
The Bloomberg Ribbon inside Excel gets updated with the monthly Bloomberg Terminal software upgrade and if the user needs to manually update this go in to http://www.bloomberg.com/professional/software_support and click the download link under Upgrade Existing.

When the user initially launches the Excel Add-In Control Panel, there will be 3 API calls for clients to get information on strategy fields behind the scene.

1. *GetBrokers* – Returns a list of brokers the user is enabled for
2. *GetBrokerStrategies* – Returns a list of strategies for a given broker
3. *GetBrokerStrategyInfo* – Returns a list of broker strategy fields for given strategy

To select the broker strategy fields, check Strategy Name and Broker Field 1 to 10 under the Staging Blotter. The user only needs to select the Broker Field where there is a need to manipulate the values inside the strategy.

For example if the user just needs to modify the start time and end time on a VWAP algorithm, they can select Strategy Name and Broker Field 1 and Broker Field 2.



Once the columns are selected and created inside the staging blotter, the user can specify the field name and value per strategy in each of the Broker Field columns.

Strategy Name	Broker Field 1	Broker Field 2
VWAP	Start Time=08:00:00	End Time=14:24:30

For example, if the user prefers to just specify Start Time for VWAP strategy, the user can add Start Time=08:00:00 inside any one of the Broker Field columns.

The time needs to be specified in “**HH:MM:SS**” **format** regardless of the EQMB <GO> setting.

For detailed view into the strategies and fields that are available, please launch your ticket in EMSX.

The field names are case sensitive.

Bloomberg

Short Selling – Excel Add-In

When the user selects SHRT under Side, there are three additional fields the user has options to include. The Locate required field is typically the only required field. If selected as “Yes”, typically the broker session also requires Locate ID or Locate Broker information. This is broker dependent field. Please contact your broker for details.

Short Selling Fields	
Locate Req	Yes or No
Locate Broker	Enter if applicable.
Locate ID	Enter if applicable.

EMSX API Excel Add-In Fields

The following fields are always populated on your staging blotter.

Staging Blotter Mandatory Fields	
Ticker	Include the Bloomberg security identifier. For example, “AAPL US Equity”
Side	BUY, SELL, SHRT (Short Sell), B/C (Buy to Close), B/O (Buy to Open), S/O, and S/C.
Order Type	MKT (Market Order) , LMT (Limit Order) and etc.
Amount	Quantity
TIF	Time in force values: DAY, GTC (Good till Cancel), GTD (Good till date) and etc.
Broker	Unique broker code used to identify the receiving broker.
Handling Instruction	Broker specified handling instruction of the order. For example , ANY, DOT, AUTO, MAN and etc.

The Request Status will display the status information.

Action Trigger	Action	Request Status	Sequence Number	Route Number
TRUE	Route	Request Processed - Order created and routed; OrderNumber = 2491636; RouteNumber = 1		
		Enter data for new request in this row ...		

Following are additional fields checked on with the default fields

Additional Fields	
LmtPrc	Limit Price of an order.
GTD Date	Good until date for GTD order type. GTD time in force allows user to select an expiration date of an order.
Stop Price	Stop Price is the predetermined entry or exit price where stop order becomes a market order.

Bloomberg

Following are the default fields you must have on your order and route blotter.

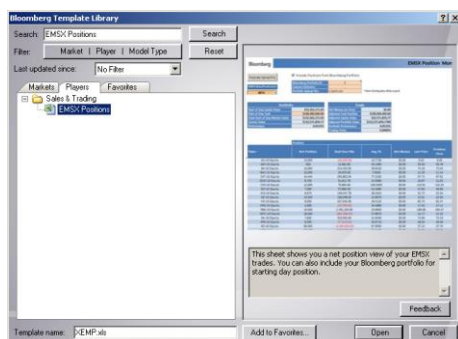
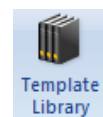
Staging Blotter Mandatory Fields	
Ticker	Include the Bloomberg security identifier. For example, "AAPL US Equity"
Side	BUY, SELL, SHRT (Short Sell), SHRX (Short Sell Exempt), BUYM (Buy Minus), COVR (Cover), B/C (Buy to Close), B/O (Buy to Open), S/O, and S/C.
Order Type	MKT (Market Order) , LMT (Limit Order) and etc.
Amount	Quantity
Working	Working quantity of an order
Rembal	Remaining quantity of an order
TIF	Time in force values: DAY, GTC (Good till Cancel), GTD (Good till date) and etc.
Broker	Unique broker code used to identify the receiving broker.
EMSX Status	Status of an order or high lights any errors.
Handling Instruction	Broker specified handling instruction of the order. For example , ANY, DOT, AUTO, MAN and etc.

Following are additional fields checked on with the default fields

Additional Fields	
AvgPrc	Average Price
LmtPrc	Limit Price
Trader	Trader name
GTD Date	Good until date for GTD order type. GTD time in force allows user to select an expiration date of an order.
Stop Price	Stop Price is the predetermined entry or exit price where stop order becomes a market order.

EMSX API Excel Add-In Use Cases

Inside the Bloomberg Ribbon, the user can select Template Library and search for EMSX specific templates. Technically anything related to Equities, Futures and Options could be used alongside the EMSX Add-In. The user can also access the templates in the Bloomberg Terminal by typing **XLTP <GO>**.

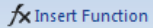


EMSX Positions template (XEMP.xls) and OSA up loader (XPUP.xls) leverages **PRTU<GO>**. **PRTU<GO>** function allows users to create, manage and share portfolios inside the Bloomberg. EMSX Positions template (XEMP.xls) can pull down the start of the day portfolio from **PRTU<GO>** and can integrate current day's transaction in EMSX for live P&L, net positions and average price on the portfolio holdings in real-time. **PORT <GO>** and or **OSA<GO>** can also be used for basic portfolio analytics and various options scenario analysis.

Bloomberg

Desktop API 3.0

The user can use various functions provided by the Desktop API 3.0 when building their custom application in Excel.

Insert Function  will display a pop-up describing various functions that the user can call using the desktop API. For example, BDP (Bloomberg Data Point), BDS (Bloomberg Data Set) and BDH (Bloomberg Data History) functions.

When building custom applications using Microsoft Excel, it is important to keep **Decimal Precision** in check.

For example, EMSX Excel Add-In will send out the decimal precision sent by the Excel to the broker. If the user sends out 4 decimal precision for a limit price, the EMSX API will send to the broker with 4 decimal precision. If the broker only supports 2 decimal precision, this order will be rejected by the broker.

DAPI<GO>

Details about the Bloomberg Excel Add-In can be found in **DAPI<GO>** in the terminal. The product overview section covers various components inside the Excel Add-In that users can use to build their model inside the Excel.

Technical Analysis

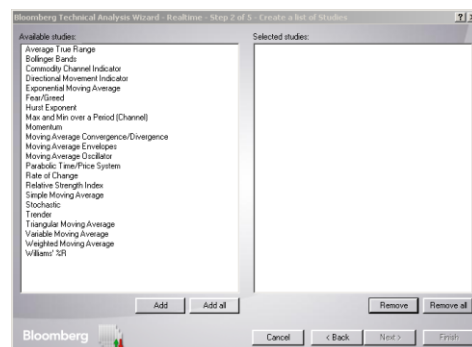
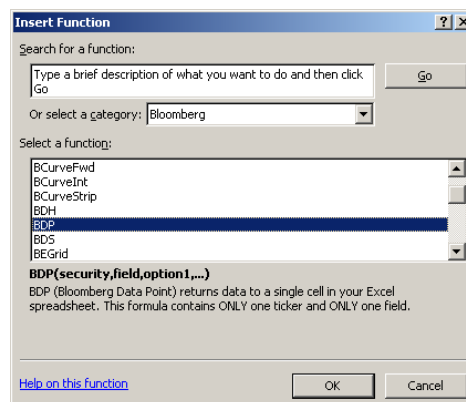
The API provides Technical Analysis studies to enable the evaluation of securities through statistical analysis.

The Technical Analysis Wizard pulls the real-time, historical, or intraday data points using BTH () and BTP () functions.

Once the user select the data type (Real Time, Historical, or Intraday) the wizard will allow the user to select the securities.

Once the securities have been selected, the available studies will show up in the next window and the user can select and make necessary changes before the Technical Studies will be populated in the Excel worksheet.

Once the information is present in the worksheet, the user can use that particular worksheet and create the necessary triggers or custom workflows to interact with the Excel Add-In Staging Blotter to send orders and routes (placements) to the market.



Bloomberg

Pairs Blotter

One common usages of the Excel Add-In is to build medium latency, medium frequency trading application. This is by no means a tutorial on building a pairs application, rather it is an example of how Excel Add-In can be used alongside the Desktop API to build various components for a client driven custom trading solutions.

The EMSX API Excel Add-In does not technically restrict user from building anything using Excel and VBA. The Desktop API also becomes a powerful complement when building these custom solutions where there is a need for real-time or static data sets. Building a Pairs blotter can be accomplished by creating in-cell formulas using various BDP functions.

I2 fx =BDP(\$C2,I\$1)

B	C	D	E	F	G	H	I
Side Symbol	Name	Note Field	Broker Code	Order Type	PX_CLOSE_1D	BID	
Buy	XOM US Equity	EXXON MOBIL CORP		MKT	86.08	86.26	
Sell	CVX US Equity	CHEVRON CORP		LMT	106.2	106.52	

The user can create a single line view of the pair and create a Pair Name.

J2 fx =D2/H2

1	A	B	C	D	E	F	G	H	I	J	K
1	Pair Name	LAST_PRICE	BID	ASK	LAST_PRICE	BID	ASK	Ratio	Leg limit		
2	JPMCB	JPM US Equity	42.98	42.94	42.95	C US Equity	33.045	33.03	33.03	1.3000	1.3200
3	CJPMB	C US Equity	33.045	33.03	33.03	JPM US Equity	42.98	42.94	42.95	0.7692	0.7892
4	JPMCS	JPM US Equity	42.98	42.94	42.95	C US Equity	33.045	33.03	33.03	0.3003	0.3203
5	CJPMS	C US Equity	33.045	33.03	33.03	JPM US Equity	42.98	42.94	42.95	-0.2310	-0.2110

Alternatively, the user can create a blotter having two line views for each pairs and create various ratios to monitor and set the trigger point to execute the pairs.

M2 fx =J2/I3

B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Side Symbol	Name	Note Field	Broker Code	Order Type	PX_CLOSE_1D	BID	ASK	LAST_PRICE	Volume	Setup Ratio	Ratio	Unwind Ratio	Spread
2	Buy	XOM US Equity	EXXON MOBIL CORP		MKT	86.08	86.26	86.27	86.34	11735103	0.8099	0.8102	0.8097	
3	Sell	CVX US Equity	CHEVRON CORP		LMT	106.2	106.52	106.53	106.56	4726927				
4	Buy	AAPL US Equity	APPLE INC			603	584.03	584.14	583.98	18046154		41.4206		
5	Sell	RIM CN Equity	RESEARCH IN MOTION			13.79	14.1	14.14	14.13	3599540				
6	Buy													
7	Sell													

When interacting with the staging blotter, the user needs to make sure to program the additional logic to accommodate situations where one leg is filled over the other and put in risk on the amount of leg risk the user is willing to accept.

Bloomberg

Building Triggers

The user can also use the Excel Add-In to build a trigger by creating a custom UDF's (User Defined Functions) using VBA code behind the scene.

Within the VBA module user can use either offset from a cell or by guiding directly to the rows and columns.

```
' Use Offset to specify
Option Explicit

Public Const POS_SHEET_STATUS_START_CELL As String = "StatusStart"
Public Const POS_SHEET_TICKER_START_CELL As String = "TickerStart"
Private Const POS_SHEET_NEW_ORDER_START_CELL As String = "TickerStart"
Private Const POS_SHEET_NEW_ORDER_QTY_COL_OFFSET As Long = 6 'Quantity
Private Const POS_SHEET_NEW_ORDER_SIDE_COL_OFFSET As Long = 7 'Buy/Sell
Private Const POS_SHEET_NEW_ORDER_TYPE_COL_OFFSET As Long = 8 'Limit/Market
Private Const POS_SHEET_NEW_ORDER_LMTPRC_COL_OFFSET As Long = 9 'Limit Price
```

```
' Specify actual columns as start cell
Option Explicit

Private Const POS_SHEET_NEW_ORDER_START_CELL As String = "TickerStart"
Private Const POS_SHEET_NEW_ORDER_QTY_START_CELL As String = "AB16" 'Quantity
Private Const POS_SHEET_NEW_ORDER_SIDE_START_CELL As String = "AF16" 'Buy/Sell
Private Const POS_SHEET_NEW_ORDER_TYPE_START_CELL As String = "AG16" 'LMT/MKT
Private Const POS_SHEET_NEW_ORDER_LMTPRC_START_CELL As String = 9 'LMT Price
```

Add new orders to the Staging Blotter.

```
Private Function AddNewOrder(posSheet As Worksheet, rowNum As Integer, _
    stageBlotterRow As Range)

    stageBlotterRow.Cells(1, STAGE_SHEET_ACTION_COL_OFFSET + 1).Value = "Route"

    stageBlotterRow.Cells(1, STAGE_SHEET_TICKER_COL_OFFSET + 1).Value = _
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET(rowNum - 1).Value

    stageBlotterRow.Cells(1, STAGE_SHEET_SIDE_COL_OFFSET + 1).Value = _
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _
            (rowNum - 1, POS_SHEET_NEW_ORDER_SIDE_COL_OFFSET).Value

    stageBlotterRow.Cells(1, STAGE_SHEET_AMOUNT_COL_OFFSET + 1).Value = _
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _
            (rowNum - 1, POS_SHEET_NEW_ORDER_QTY_COL_OFFSET).Value

    stageBlotterRow.Cells(1, STAGE_SHEET_ORDER_TYPE_COL_OFFSET + 1).Value = _
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _
            (rowNum - 1, POS_SHEET_NEW_ORDER_TYPE_COL_OFFSET).Value
```

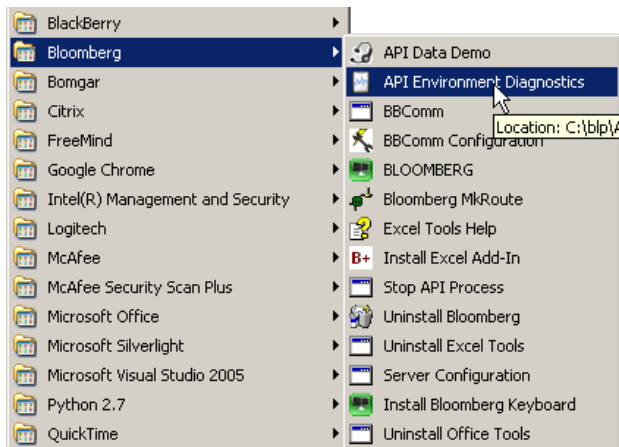

Bloomberg

```
If "LMT" = UCase _  
    (Trim(stageBlotterRow.Cells(1, STAGE_SHEET_ORDER_TYPE_COL_OFFSET + _  
        1).Value))  
Then  
    stageBlotterRow.Cells(1, STAGE_SHEET_LIMIT_PRC_COL_OFFSET + 1).Value = _  
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
            (rowNum - 1, POS_SHEET_NEW_ORDER_LMTPRC_COL_OFFSET).Value  
End If  
  
    stageBlotterRow.Cells(1, STAGE_SHEET_TIF_COL_OFFSET + 1).Value = _  
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
            (rowNum - 1, POS_SHEET_NEW_ORDER_TIF_COL_OFFSET).Value  
  
If "GTD" = posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
    (rowNum - 1, POS_SHEET_NEW_ORDER_TIF_COL_OFFSET).Value  
  
Then stageBlotterRow.Cells _  
    (1, STAGE_SHEET_GTD_DATE_COL_OFFSET + 1).Value _  
    = posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
        (rowNum - 1, POS_SHEET_NEW_ORDER_GTD_COL_OFFSET).Value  
  
Else  
    stageBlotterRow.Cells _  
        (1, STAGE_SHEET_GTD_DATE_COL_OFFSET + 1).ClearContents  
  
End If  
  
    stageBlotterRow.Cells(1, STAGE_SHEET_BROKER_COL_OFFSET + 1).Value = _  
        posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
            (rowNum - 1, POS_SHEET_NEW_ORDER_BROKER_COL_OFFSET).Value  
  
    stageBlotterRow.Cells(1, STAGE_SHEET_HANDL_INSTR_COL_OFFSET + 1).Value _  
        = posSheet.Range(POS_SHEET_NEW_ORDER_START_CELL).OFFSET _  
            (rowNum - 1, POS_SHEET_NEW_ORDER_HANDL_INSTR_COL_OFFSET).Value  
  
End Function
```

Bloomberg

Troubleshooting EMSX API Excel Add-In

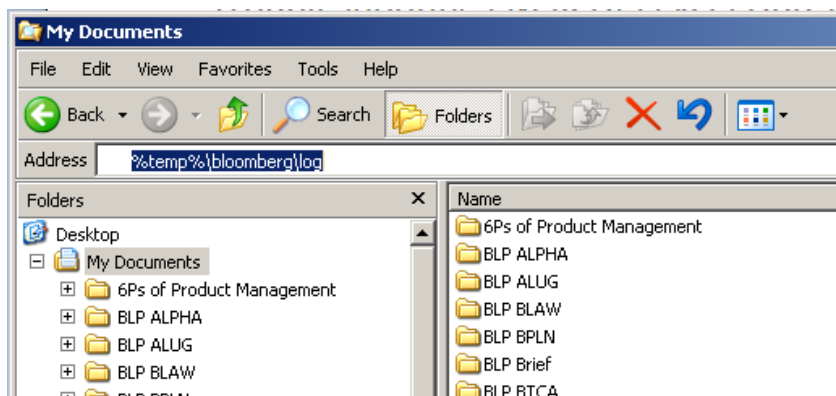
Most issues that arise from Excel can be resolved with the Bloomberg API Diagnostics Tool.



The diagnostic tool will run diagnostic check and self-repair by clicking on the Repair button once the initial diagnostics are complete. This tool can be found inside.

- Start
- All Programs
- Bloomberg
- API Environment Diagnostics

From time to time EMSX Support may also ask for the EMSX Engine log to troubleshoot the error. To grab the latest log for the EMSX API Excel Add-In, the user needs to type **%temp%\bloomberg\log** in the address bar section of windows explorer. Within this folder, the user will see the “EMSXEngine_MM_DD_YYYY.log”file.



Bloomberg

To upgrade to the latest version of the software go to http://www.bloomberg.com/professional/software_support and click the download link under Upgrade Existing.

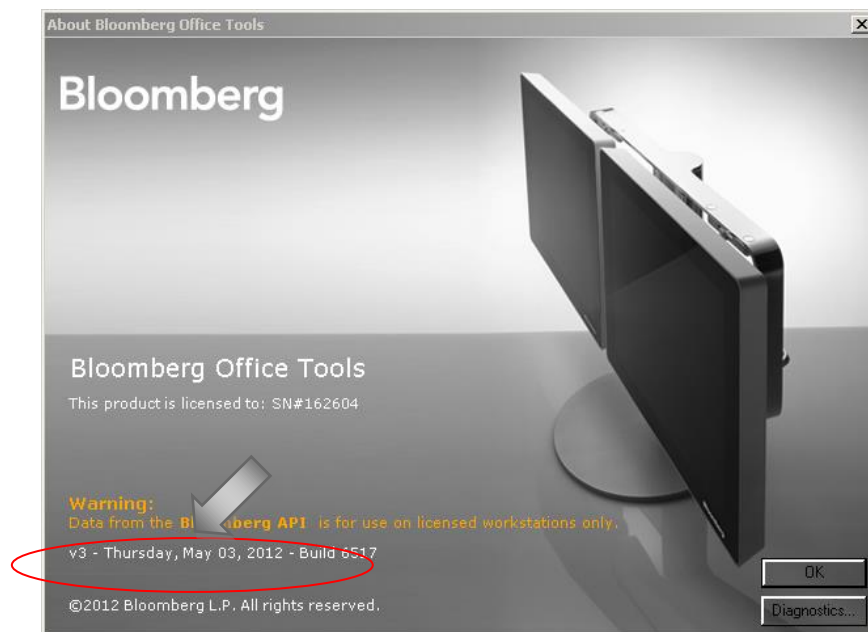
Client can also go to **UPGR<GO>** to check if the Excel Add-in and BBComm are up to date.

User can check the software version by clicking Live Help and About.



This will bring up the splash screen that shows the date and the build number of the Excel Add-In.

For example, v3 – Thursday, May 03, 2012 – Build 6517



Bloomberg

EMSX API – COM (Component Object Model)

The Bloomberg Desktop COM v3 Data Control can also be used to access EMSX API.

The COM interface is an abstract interface that allows binary codes to be shared upon different applications and programming languages. Python is becoming one of the popular programming languages of choice by many programmers to control Excel using COM outside of the VBA.

In the COM sample codes, the *clsEMSX*, *clsEMSXSendOrd* and *clsEMSXSendOrdWithStrat* class modules are the ones you want to focus on.

Class Modules

clsEMSX	Under ‘Public Sub Subscribe’, the user needs to specify the each parameters user wants to return in the topic string.
---------	---

‘COM sample in Visual Basic

```
Public Sub Subscribe()
    Dim topicStr As String

    'Elements for some of the Route fields
    topicStr = "//blp/" & BETA_PROD_ID &
        "/route?fields=EMSX_FILL_ID,EMSX_LAST_MARKET,EMSX_LAST_PRICE,EMSX_LAST_SHARES, _
        EMSX_LAST_FILL_TIME,EMSX_LAST_FILL_DATE,EMSX_STATUS"

    Dim cid As blpapicomLib2.CorrelationId
    Set cid = m_BBG_EMSX.CreateCorrelationId(1)

    Set m_subs = m_BBG_EMSX.CreateSubscriptionList()
    m_subs.AddEx topicStr, , , cid

    ' Unsubscribe first before making new subscriptions
    Unsubscribe
    m_BBG_EMSX.Subscribe m_subs

End Sub
```

Class Modules

clsEMSXSendOrd	Under ‘Public Sub SendOrder’ the user can specify the parameters to be included.
----------------	--

‘COM sample in Visual Basic

```
Public Sub SendOrder()

    ' The request objects
    Dim req As REQUEST
```

Bloomberg

```

Dim nRow As Long
Dim service As service

Set service = m_BBG_EMSX.GetService("//blp/emapisvc_beta")

Set req = service.CreateRequest("CreateOrderAndRoute")

' Send the request m_BBG_EMSX.SendRequest req

req.GetElement("EMSX_BROKER").SetValue ("BMTB")
req.GetElement("EMSX_SIDE").SetValue ("SELL")
req.GetElement("EMSX_TICKER").SetValue ("XOM US Equity")
req.GetElement("EMSX_TIF").SetValue ("DAY")
req.GetElement("EMSX_ORDER_TYPE").SetValue ("MKT")
req.GetElement("EMSX_LOCATE_REQ").SetValue ("N")
req.GetElement("EMSX_LOCATE_BROKER").SetValue ("UBS")
req.GetElement("EMSX_AMOUNT").SetValue (1000)
req.GetElement("EMSX_HAND_INSTRUCTION").SetValue ("ANY")
req.GetElement("EMSX_RELEASE_TIME").SetValue (-1)

' Send the request
m_BBG_EMSX.SendRequest req

End Sub

```

Class Modules

clsEMSXSendOrdWithStrat	Under 'Public Sub SendOrderWithStrat' the user can specify both the order parameters and broker strategy parameters.
-------------------------	--

```

'COM sample in Visual Basic

Public Sub SendOrderWithStrat( side As String)

' The request objects
Dim req As REQUEST
Dim nRow As Long
Dim service As service
Dim strategy As Element
Dim indicator As Element
Dim strategy_data As Element

' Get service
Set service = m_BBG_EMSX.GetService("//blp/emapisvc_beta")

' Create request object
Set req = service.CreateRequest("CreateOrderAndRouteWithStrat")

' Security and field elements
req.GetElement("EMSX_BROKER").SetValue ("BMTB")
req.GetElement("EMSX_SIDE").SetValue ("SELL")
req.GetElement("EMSX_TICKER").SetValue ("AAPL US Equity")
req.GetElement("EMSX_TIF").SetValue ("DAY")
req.GetElement("EMSX_ORDER_TYPE").SetValue ("MKT")
req.GetElement("EMSX_LOCATE_REQ").SetValue ("N")

```

Bloomberg

```
req.GetElement("EMSX_LOCATE_BROKER").SetValue ("UBS")
req.GetElement("EMSX_AMOUNT").SetValue (1000)
req.GetElement("EMSX_HAND_INSTRUCTION").SetValue ("ANY")
req.GetElement("EMSX_RELEASE_TIME").SetValue (-1)

Set strategy = req.GetElement("EMSX_STRATEGY_PARAMS")
  strategy.GetElement("EMSX_STRATEGY_NAME").SetValue _
    Range("SELECTED_STRAT").Value

Set indicator = strategy.GetElement("EMSX_STRATEGY_FIELD_INDICATORS")
Set strategy_data = strategy.GetElement("EMSX_STRATEGY_FIELDS")

Dim indicator_Data As Element
Dim stratData As Element
Set indicator_Data = indicator.AppendElement()
Set stratData = strategy_data.AppendElement()

Dim i As Integer
Dim rowCounter As Integer, colCounter As Integer, totCounter As Integer
Dim fieldInd As Integer, fieldData As Variant

For i=0 to Range("NUM_PARAMS").Value - 1
  fieldInd = shtStrategy.Range("G13").Offset(totCounter + _
    rowCounter, colCounter).Value
  fieldData = shtStrategy.Range("G14").Offset(totCounter + _
    rowCounter, colCounter).Value

  indicator_Data.SetElement "EMSX_FIELD_INDICATOR", fieldInd
  stratData.SetElement "EMSX_FIELD_DATA", fieldData

  If (i+1) Mod 4 <> 0 Then
    rowCounter = 0
    colCounter = colCounter + 3
  Else
    totCounter = i+1
    rowCounter = 0
    colCounter = 0
  End If
Next i

' Send the request
m_BBG_EMSX.SendRequest req

End Sub
```

Schema for Request / Response Messages

Please run the attached `emapisvc_schema.html` file.