Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The copyright notice below and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

BLOOMBERG is a registered trademark of Bloomberg Finance L.P. or its affiliates.

All other trademarks and registered trademarks are the property of their respective owners.
# Table of Contents

Preface: About this Document .......................................................................................................... 9  
Purpose ............................................................................................................................................... 9  
Audience ............................................................................................................................................. 9  
Document History ............................................................................................................................... 9  
Customer Support Information ......................................................................................................... 10  
1 Introduction to the Bloomberg API ............................................................................................... 12  
1.1 Overview of the Bloomberg API .............................................................................................. 12  
1.1.1 Features ................................................................................................................................ 13  
1.1.2 The Bloomberg Platform ...................................................................................................... 15  
1.1.3 B-PIPE ...................................................................................................................................... 16  
1.1.4 The Desktop API and Server API ......................................................................................... 17  
1.2 API Typical Application Structure ............................................................................................ 20  
1.3 Overview of this Guide ............................................................................................................... 21  
2 Sample Programs in Two Paradigms ............................................................................................. 22  
2.1 Overview ..................................................................................................................................... 22  
2.2 The Two Paradigms .................................................................................................................... 23  
2.2.1 Request/Response .................................................................................................................. 23  
2.2.2 Subscription .......................................................................................................................... 24  
2.3 Using the Request/Response Paradigm ....................................................................................... 24  
2.4 Using the Subscription Paradigm ............................................................................................... 28  
3 Sessions and Services ..................................................................................................................... 31  
3.1 Sessions ...................................................................................................................................... 31  
3.2 Services ...................................................................................................................................... 31  
3.3 Event Handling ........................................................................................................................... 31  
3.3.1 Synchronous Event Handling .............................................................................................. 33  
3.3.2 Asynchronous Event Handling ............................................................................................ 34  
3.4 Multiple Sessions ....................................................................................................................... 38  
4 Requests and Responses ................................................................................................................ 39  
4.1 The Programming Example ...................................................................................................... 39  
4.2 Elements ..................................................................................................................................... 40  
4.3 Request Details ........................................................................................................................... 40  
4.4 Response Details ......................................................................................................................... 42  
5 Subscriptions ................................................................................................................................. 47  
5.1 Starting a Subscription ................................................................................................................ 47
5.2 Receiving Data from a Subscription ................................................................. 49
5.3 Modifying an Existing Subscription ................................................................. 50
5.4 Stopping a Subscription ...................................................................................... 51
5.5 Overlapping Subscriptions .................................................................................. 51
5.6 Conflation and the Interval Option ....................................................................... 52
5.7 Delayed Data ......................................................................................................... 52
5.8 Subscription Life Cycle ......................................................................................... 52

6 Authorization and Permissioning Systems ............................................................ 54
6.1 Overview .............................................................................................................. 54
6.2 Underlying Concepts ......................................................................................... 54
   6.2.1 EIDs .............................................................................................................. 54
   6.2.2 Requirement for the Terminal ....................................................................... 54
   6.2.3 The //blp/apiauth service .......................................................................... 55
   6.2.4 The V3 Identity Object .............................................................................. 55
   6.2.5 V3 Permissioning Models .......................................................................... 55
   6.2.6 Authorization Lifetime ............................................................................... 55
6.3 Server API Authorization ..................................................................................... 56
   6.3.1 Authorization by IP Address ....................................................................... 56
6.4 B-PIPE Authorization ......................................................................................... 62
   6.4.1 Authentication ............................................................................................. 63
   6.4.2 Token Generation ....................................................................................... 65
   6.4.3 Identity Object ............................................................................................ 67
6.5 Authorization ....................................................................................................... 67
6.6 Permissioning ...................................................................................................... 69
   6.6.1 Entitlements .................................................................................................. 69
   6.6.2 User Mode ................................................................................................... 72
   6.6.3 Content Based ............................................................................................ 72
6.7 Specific Application Types (B-PIPE only) ............................................................ 74
   6.7.1 Single-User ................................................................................................ 74
   6.7.2 Multi-User .................................................................................................. 74
   6.7.3 Derived Data / Non-Display ....................................................................... 74
6.8 V2 Authorization and Permissioning Models ....................................................... 74
   6.8.1 User Mode ................................................................................................ 74
   6.8.2 All-or-None ................................................................................................ 75
   6.8.3 Content-Based / Per-Product / Per-Security ............................................. 75
   6.8.4 Validating Logon Status ........................................................................... 76
# Core Services

7.1 Common Concepts

7.1.1 Security/Securities

7.1.2 Pricing Source

7.1.3 Fields

7.1.4 Overrides

7.1.5 Relative Dates

7.2 Reference Data Service

7.2.1 Reference Data Request and Response Overview

7.2.2 Historical Data Request

7.2.3 Intraday Tick Request

7.2.4 Intraday Bar Services

7.2.5 Portfolio Data Request

7.2.6 BEQS Request

7.3 Market Data Service

7.4 Custom VWAP Service

7.5 Market Bar Subscription Service

7.6 API Field Information Service

7.6.1 Field Information Request

7.6.2 Field Search Request

7.6.3 Categorized Field Search Request

7.7 Page Data Service

7.8 Technical Analysis Service

7.8.1 Historical End of Day study request

7.8.2 Intraday bar study request

7.8.3 Real time study request

7.9 API Authorization

7.10 Instruments Service

7.10.1 Security Lookup Request

7.10.2 Curve Lookup Request

7.10.3 Government Lookup Request

7.10.4 Response Behaviors

7.10.5 Code Example

# Publishing

8.1 Overview

8.2 The Programming Examples
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3.4 Field Search Request</td>
<td>187</td>
</tr>
<tr>
<td>A.3.5 Categorized Field Search Request</td>
<td>191</td>
</tr>
<tr>
<td>A.3.6 Field List Request</td>
<td>194</td>
</tr>
<tr>
<td>A.3.7 Field Service Response Elements</td>
<td>196</td>
</tr>
<tr>
<td>A.3.8 Field Service Response Values</td>
<td>197</td>
</tr>
<tr>
<td>A.4 Market Bar Subscription</td>
<td>198</td>
</tr>
<tr>
<td>A.4.1 Market Bar Subscription Settings</td>
<td>198</td>
</tr>
<tr>
<td>A.4.2 Market Bar Subscription: Data Events Response</td>
<td>198</td>
</tr>
<tr>
<td>A.5 Schema for Market Data and Custom VWAP</td>
<td>201</td>
</tr>
<tr>
<td>A.5.1 MarketDataEvents: Choice</td>
<td>201</td>
</tr>
<tr>
<td>A.5.2 Market Data Service Subscription Options</td>
<td>201</td>
</tr>
<tr>
<td>A.5.3 MarketDataEvents: Sequence</td>
<td>201</td>
</tr>
<tr>
<td>A.5.4 Market VWAP Service Subscription Options</td>
<td>212</td>
</tr>
<tr>
<td>A.6 Schema for API Authorization</td>
<td>213</td>
</tr>
<tr>
<td>A.6.1 Authorization Request</td>
<td>213</td>
</tr>
<tr>
<td>A.6.2 Authorization Request Response</td>
<td>214</td>
</tr>
<tr>
<td>A.6.3 Logon Status Request</td>
<td>215</td>
</tr>
<tr>
<td>A.6.4 Logon Status Request Response</td>
<td>215</td>
</tr>
<tr>
<td>A.6.5 User Entitlements Request</td>
<td>216</td>
</tr>
<tr>
<td>A.6.6 User Entitlements Request Response</td>
<td>216</td>
</tr>
<tr>
<td>A.6.7 Security Entitlements Request</td>
<td>217</td>
</tr>
<tr>
<td>A.6.8 Security Entitlements Request Response</td>
<td>217</td>
</tr>
<tr>
<td>A.6.9 Authorization Token Request</td>
<td>218</td>
</tr>
<tr>
<td>A.6.10 Authorization Token Request Response</td>
<td>218</td>
</tr>
<tr>
<td>A.6.11 Field Service Response Values</td>
<td>219</td>
</tr>
<tr>
<td>A.6.12 Field Service Request Values</td>
<td>219</td>
</tr>
<tr>
<td>B Error Codes</td>
<td>220</td>
</tr>
<tr>
<td>B.1 General</td>
<td>220</td>
</tr>
<tr>
<td>B.2 //BLP/APIAUTH</td>
<td>221</td>
</tr>
<tr>
<td>B.2.1 AUTHORIZATION_STATUS, REQUEST_STATUS, RESPONSE and PARTIAL_RESPONSE Events</td>
<td>221</td>
</tr>
<tr>
<td>B.2.2 REQUEST_STATUS, RESPONSE and PARTIAL_RESPONSE Events (B-Pipe ONLY)</td>
<td>223</td>
</tr>
<tr>
<td>B.2.3 TOKEN_STATUS Event (B-Pipe ONLY)</td>
<td>224</td>
</tr>
<tr>
<td>B.3 //BLP/MKTDATA and //BLP/MKT VWAP</td>
<td>224</td>
</tr>
<tr>
<td>B.3.1 (SUBSCRIPTION_DATA and SUBSCRIPTION_STATUS Events)</td>
<td>224</td>
</tr>
</tbody>
</table>
Table of Contents

Bloomberg

F C Examples ................................................................................................................................. 312
  F.1 RequestResponseParadigm ........................................................................................ 313
  F.2 Subscription Paradigm ............................................................................................ 318
  F.3 Asynchronous Event Handling .................................................................................. 327
  F.4 Request Response Multiple ...................................................................................... 332
  F.5 Subscription Multiple .............................................................................................. 340
Preface: About this Document

Purpose

This document provides a guide to developing applications using the Bloomberg API.

Audience

This document is intended for developers who use the Bloomberg API.

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>11/05/09</td>
<td>This is the first release of the Bloomberg API Developer’s Guide.</td>
</tr>
<tr>
<td>2.41</td>
<td>10/03/12</td>
<td>Corrected items in Table 9-4, “Chain Subservice Examples,” on page 142 and Table 9-4, “Chain Subservice Examples,” on page 142.</td>
</tr>
<tr>
<td>2.42</td>
<td>11/14/12</td>
<td>Updated “IntradayTickResponse: Choice” on page 173.</td>
</tr>
<tr>
<td>2.43</td>
<td>12/21/12</td>
<td>Updated “IntradayBarRequest: Sequence” on page 175.</td>
</tr>
<tr>
<td>2.44</td>
<td>01/04/13</td>
<td>Added footnote to Table 9-14, “Enumeration Values,” on page 156 and updated Table 9-4, “Chain Subservice Examples,” on page 142.</td>
</tr>
<tr>
<td>2.45</td>
<td>01/14/13</td>
<td>Updated “B-Pipe” on page 117.</td>
</tr>
<tr>
<td>2.46</td>
<td>01/29/13</td>
<td>Added “Instruments Service” on page 101. Updated MD_BOOK_TYPE table on page 125.</td>
</tr>
<tr>
<td>2.47</td>
<td>03/21/13</td>
<td>Updated MD_BOOK_TYPE table on page 125 and Notes on page 133.</td>
</tr>
<tr>
<td>2.48</td>
<td>06/05/13</td>
<td>Product name change from Managed B-PIPE to B-PIPE.</td>
</tr>
<tr>
<td>2.49</td>
<td>07/10/13</td>
<td>Fixed a typo on page 77 (comdy to comdty).</td>
</tr>
<tr>
<td>2.50</td>
<td>01/21/14</td>
<td>Updated fields in Table A.5.3, “MarketDataEvents: Sequence,” on page 201.</td>
</tr>
<tr>
<td>2.51</td>
<td>04/16/14</td>
<td>Added “Error Codes” on page 220.</td>
</tr>
<tr>
<td>2.5</td>
<td>04/17/14</td>
<td>Updated “Intraday Tick Request” on page 84.</td>
</tr>
<tr>
<td>2.53</td>
<td>05/12/14</td>
<td>Updated “REQUEST_STATUS, RESPONSE and PARTIAL_RESPONSE Events (B-Pipe ONLY)” on page 223.</td>
</tr>
<tr>
<td>2.54</td>
<td>06/30/14</td>
<td>Updated “Market Bar Subscription Service” on page 88, “Market Bar Subscription” on page 198 and Table 9-4 on page 142.</td>
</tr>
</tbody>
</table>
Customer Support Information

Urgent and Operational Support

For any urgent operational issues contact the Production Support team. Please have the following information available:

- Firm Name
- For B-PIPE the BPID/BMDS instance(s) impacted
- For Server API the ASID number
- Issue description
  - Time issue occurred
  - Error messages
  - Supporting information, such as, example securities and data
  - SDK logs (if possible)
- Contact information
  - Client name/E-mail address/Phone numbers

You can reach the Production Support team at:

- **Americas:** +1-212-617-4390
- **Europe:** +44-20-3216-4380
- **Japan:** +81-3 3201-2780
- **Hong Kong:** +852-2293-1238
- **Singapore:** +65 6212-1180
- **Australia:** +612-9777-7210

If you are a Server API user, please have your ASID number and ASID Serial Number ready when requesting support. You can find this information in the `bin/clientid.txt` file (located in the root directory that you specified as part of the Server API installation procedure).

Server API Related Questions

Press the **HELP** key twice on a Bloomberg keyboard.

Press **F1** twice on a standard keyboard.

If you are a Server API user, the first line of your request should state that you are a Server API user and include your ASID number to ensure that your request is routed quickly and correctly.
B-PIPE Related Questions

B-PIPE FAQ

The B-PIPE is available at https://software.bloomberg.com/BPIPE/sub/docs/faq.pdf

FTP and Web Site

Current B-PIPE documentation, errata, notices, data content information and the SDK are available on the B-PIPE web site, https://software.bloomberg.com/BPIPE

Non-Urgent Support

Submit a non-urgent request at: https://software.bloomberg.com/BPIPE/sub1/dlwp/b?action=PostQuery

Sales Support

Call your Bloomberg sales representative.
1 Introduction to the Bloomberg API

1.1 Overview of the Bloomberg API

The Bloomberg API provides developers with 24x7 programmatic access to data from the Bloomberg Data Center for use in customer applications.

The Bloomberg API lets you integrate streaming real-time and delayed data, reference data, historical data, intraday data, and Bloomberg-derived data into your own custom and third-party applications. You can choose which data you require down to the level of individual fields.

The Bloomberg API uses an event-driven model. The interface is thread-safe and thread-aware, giving applications the ability to utilize multiple processors efficiently. The Bloomberg API automatically breaks large results into smaller chunks and can provide conflated streaming data to improve bandwidth usage and the latency of applications.

The Bloomberg API supports run-time downloadable schemas for the services it provides, and it provides methods to query these schemas at runtime. This means the Bloomberg API can support additional services without additions to the interface. It also makes writing applications that can adapt to changes in services or entirely new services simple.
### Features

<table>
<thead>
<tr>
<th><strong>Feature</strong></th>
<th><strong>Details</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Four Languages, One Interface</td>
<td>API 3.0 provides all new programming interfaces in:</td>
</tr>
<tr>
<td></td>
<td>- Java</td>
</tr>
<tr>
<td></td>
<td>- C</td>
</tr>
<tr>
<td></td>
<td>- C++</td>
</tr>
<tr>
<td></td>
<td>- .Net</td>
</tr>
<tr>
<td></td>
<td>The Java, .Net and C++ object models are identical, while the C interface provides a C-style version of the object model. You are able to effortlessly port applications among these languages as the needs of your applications change.</td>
</tr>
<tr>
<td>Lightweight Interfaces</td>
<td>The API 3.0 programming interface implementations are extremely lightweight. The lightweight design makes the process of receiving data from Bloomberg and delivering it to applications as efficient as possible.</td>
</tr>
<tr>
<td></td>
<td>It is now possible to get the maximum performance out of the Java, .Net, C, and C++ versions of the interface.</td>
</tr>
<tr>
<td>Extensible Service-Oriented Data Model</td>
<td>The new API generically understands the notions of subscription and request-response services.</td>
</tr>
<tr>
<td></td>
<td>The subscribe method and request method allow you to send requests to different data services with potentially different or overlapping data dictionaries and different response schemas.</td>
</tr>
<tr>
<td></td>
<td>This, in combination with the new canonical data form, means that Bloomberg can deliver new data services via the API without having to extend the interface to support the new services.</td>
</tr>
<tr>
<td>Field Level Subscriptions</td>
<td>You are now able to request updates for only the fields of interest to your application, rather than receiving all trade and quote fields when you establish a subscription.</td>
</tr>
<tr>
<td></td>
<td>This reduces the overhead of processing unwanted data within both the API and your application, and also reduces network bandwidth consumption between Bloomberg and its customers.</td>
</tr>
<tr>
<td></td>
<td>For example, if quotes are of no interest to an application, processing and bandwidth consumption can be cut by as much as 90%.</td>
</tr>
</tbody>
</table>
### Summary events

When you subscribe to market data for a security, the API performs two actions:

1. It retrieves a summary of the current state of the security and delivers it to you.

A summary is made up of data elements known as fields. The set of summary fields varies depending on the asset class of the requested security.

2. The API streams all market data updates to you as they occur and continues to do so until you cancel the subscription.

About 300 market data fields are available via the API subscription interface, most of them derived from trade and quote events.

### Interval-based Subscriptions

Many users of API data are interested in subscribing to large sets of streaming data but only need summaries of each requested security to be delivered at periodic intervals.

The API subscription model allows you to specify the minimum interval at which to receive streaming updates. This reduces processing and bandwidth consumption by delivering only an updated summary at the interval you define.

It is also possible to establish multiple subscriptions such that a summary arrives periodically but other fields, such as trade related fields, are delivered in real time.

### No Request Size Restrictions

API 3.0 allows you to request a potentially unlimited number of securities and fields without having to manage request rates yourself.

The API infrastructure manages the distribution of these requests across Bloomberg's back end data servers, which in turn ensure that all arriving data requests are given equal access to the available machine resources.

### Canonical Data Format

Each data field returned to an application via the API is now accompanied by an in-memory dictionary element that indicates the data type (for example, integer, double) and provides a description of the field - the data is self-describing.

Data elements may be simple, such as a price field, or complex, such as historical prices or bulk fields. All data is represented in the same canonical form and developers do not have to deal with multiple data formats or be exposed to the details of the underlying transport protocol.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary events</strong></td>
<td>When you subscribe to market data for a security, the API performs two actions:</td>
</tr>
<tr>
<td></td>
<td>1. It retrieves a summary of the current state of the security and delivers it to you.</td>
</tr>
<tr>
<td></td>
<td>A summary is made up of data elements known as fields. The set of summary fields varies depending on the asset class of the requested security.</td>
</tr>
<tr>
<td></td>
<td>2. The API streams all market data updates to you as they occur and continues to do so until you cancel the subscription.</td>
</tr>
<tr>
<td></td>
<td>About 300 market data fields are available via the API subscription interface, most of them derived from trade and quote events.</td>
</tr>
<tr>
<td><strong>Interval-based Subscriptions</strong></td>
<td>Many users of API data are interested in subscribing to large sets of streaming data but only need summaries of each requested security to be delivered at periodic intervals.</td>
</tr>
<tr>
<td></td>
<td>The API subscription model allows you to specify the minimum interval at which to receive streaming updates. This reduces processing and bandwidth consumption by delivering only an updated summary at the interval you define.</td>
</tr>
<tr>
<td></td>
<td>It is also possible to establish multiple subscriptions such that a summary arrives periodically but other fields, such as trade related fields, are delivered in real time.</td>
</tr>
<tr>
<td><strong>No Request Size Restrictions</strong></td>
<td>API 3.0 allows you to request a potentially unlimited number of securities and fields without having to manage request rates yourself.</td>
</tr>
<tr>
<td></td>
<td>The API infrastructure manages the distribution of these requests across Bloomberg's back end data servers, which in turn ensure that all arriving data requests are given equal access to the available machine resources.</td>
</tr>
<tr>
<td><strong>Canonical Data Format</strong></td>
<td>Each data field returned to an application via the API is now accompanied by an in-memory dictionary element that indicates the data type (for example, integer, double) and provides a description of the field - the data is self-describing.</td>
</tr>
<tr>
<td></td>
<td>Data elements may be simple, such as a price field, or complex, such as historical prices or bulk fields. All data is represented in the same canonical form and developers do not have to deal with multiple data formats or be exposed to the details of the underlying transport protocol.</td>
</tr>
</tbody>
</table>
The Bloomberg API is the interface to the following Bloomberg products:

- The Bloomberg Platform
- B-PIPE
- Server API
- Desktop API

1.1.2 The Bloomberg Platform

The Bloomberg Platform is a revolutionary step in market data distribution — a new managed service that extends well beyond traditional industry solutions. Providing real-time delayed, and historical market data, as well as global publishing, trusted entitlements, and much more,
the Bloomberg Platform is a complete high-volume, low-latency service to end users, applications, and displays throughout your entire financial firm (see Figure 1-1).

Figure 1-1: The Bloomberg Platform

1.1.3 B-PIPE

B-PIPE leverages the Bloomberg distribution platform and managed entitlements system. B-PIPE allows clients to connect applications providing solutions that work with client proprietary and 3rd party applications. B-PIPE provides the tools to permission data to entitled users only. Client applications will use the Bloomberg entitlements system to ensure distribution of data only to appropriately entitled users (see Figure 1-2).
1.1.4 The Desktop API and Server API

The Desktop API and Server API have the same programming interface and behave almost identically. The chief difference is that customer applications using the Server API have some additional responsibilities. Those additional requirements will be detailed later in this document (see Bloomberg API Developer’s Guide: Authorization and Permissioning); otherwise, assume the two deployments are identical.

Note that in both deployments, the end-user application and the customer’s active BLOOMBERG PROFESSIONAL service share the same display/monitor(s).
The Desktop API

The Desktop API 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 Data Center (see Figure 1-3).

The Server API

The Server API allows customer end-user applications to obtain data from the Bloomberg Data Center via a dedicated process, known as the Server API process. Introduction of the Server API process allows, in some circumstances, better use of network resources.

When the end-user applications interact directly with the Server API process they are using the Server API in User Mode (see Figure 1-4).
When the customer implements a Customer Server Application to interact with the Server API process (see Figure 1-5), the Server API is then being used in Server Mode (by the Customer Server Application). Interactions between the Customer Server Application and the Customer End-User Application(s) are handled by an application protocol of the customer’s design.
1.2 Typical Application Structure

The Bloomberg API object model contains a small number of key objects which applications use to request, receive and interpret data.

An application creates a `Session` object to manage its connection with the Bloomberg infrastructure. (Some applications may choose to create multiple `Session` objects for redundancy).

---

**Figure 1-5: The Server API: Server Mode**
Using the Session object, an application creates a Service object and then “opens’ each Bloomberg service that it will use. For example, Bloomberg provides streaming market data and reference data as services.

There are two programming paradigms that can be used with the Service object. The client can make individual requests for data (via a Request object) or the client can start a subscription with the service (managed via a Subscription object) for ongoing data updates. A customer application may be written to handle both paradigms. Whichever paradigm or paradigms are used, the Bloomberg infrastructure replies with events (received at the client as Event objects) which the client must handle asynchronously.

Programmatically, the customer application obtains Event objects for the Session and then extracts from each Event object one or more Message objects containing the Bloomberg data.

### 1.3 Overview of this Guide

The rest of this guide is arranged as follows

- First a small but complete example program is presented to illustrate the most common features of the Bloomberg API. See “Sample Programs in Two Paradigms” on page 22.
- This is followed by detailed descriptions of the key scenarios in using the Bloomberg API: creating a session; opening services; sending requests and processing their responses; and subscribing to streaming data and processing the results. See “Sessions and Services” on page 31, “Requests and Responses” on page 39, and “Subscriptions” on page 47.
2 Sample Programs in Two Paradigms

2.1 Overview

This chapter demonstrates the most common usage patterns of the Bloomberg API. The major programming issues are addressed at a high level and working example code is provided as a way to quickly get started with your own applications. Later chapters will provide additional details that are covered lightly here. The Bloomberg API has two different models for providing data (the choice usually depends on the nature of the data): request/response and subscription. Both models are shown in this chapter.

The major steps required of an application are:

- The creation and startup of a `Session` object which the application uses to specify the data it wants and then receive that data.
- Data from the Bloomberg infrastructure is organized into various "services". The application "opens" the service that can provide the needed data (e.g., reference data, current market data).
- The application asks the service for specific information of interest. For example, the last price for a specific security.
- The application waits for the data to be delivered.

Data from the service will arrive in one or more asynchronously delivered `Event` objects. If an application has several outstanding requests for different data, the data arriving from these multiple requests may be interleaved with each other; however, data related to a specific request always arrives in order.

*Note: To assist applications in matching incoming data to requests, the Bloomberg API allows applications to provide a `CorrelationID` object with each request. Subsequently, the Bloomberg infrastructure uses that identifier to tag the events sent in response. On receipt of the `Event` object, the client can use the identifier it supplied to match events to requests.*

Even if an application (such as the examples in this chapter) makes only a single request for data, the application must also be prepared to handle status events from the service in addition to the requested data.
The following display provides an outline of the organization used in these examples.

```java
import classes
public class Example1 {
    private static void handleDataEvent(Event event) throws Exception {
        ........
    }
    private static handleOtherEvent(Event event) throws Exception {
        ........
    }
    public static void main(String[] args) throws Exception {
        create and start Session
        use Session to open service
        ask service for data
            (provide id for service to label replies)
            loop waiting for data; pass replies to event handlers
    }
}
```

The additional details needed to create a working example are provided below.

### 2.2 The Two Paradigms

Before exploring the details for requesting and receiving data, we describe the two different paradigms used by the Bloomberg API - Request/Response and Subscription.

The Service defines which paradigm is used to access it. For example, the streaming real-time market data service uses the subscription paradigm whereas the reference data service uses the request/response paradigm. See "Core Services" on page 77 for more information on the Core Services provided by the Bloomberg API.

Note: Applications that make heavy use of real-time market data should use the streaming real-time market data service. However, real-time information is available through the reference data service requests where you will get a snapshot of the current value in the response.

#### 2.2.1 Request/Response

In this case, data is requested by issuing a `Request` and is returned in a sequence consisting of zero or more `Events` of type `PARTIAL_RESPONSE` followed by exactly one `Event` of type `RESPONSE`. The final `RESPONSE` indicates that the `Request` has been completed.
In general, applications written to this paradigm will perform extra processing after receiving the final response from a Request.

2.2.2 Subscription

In this case a Subscription is created which results in a stream of updates being delivered in events of type SUBSCRIPTION_DATA until the Subscription is explicitly cancelled by the application.

2.3 Using the Request/Response Paradigm

A main function for a small but complete example using the Request/Response paradigm is shown below:

```java
public static void main(String[] args) throws Exception {
    SessionOptions sessionOptions = new SessionOptions();
    sessionOptions.setServerHost("localhost"); // default value
    sessionOptions.setServerPort(8194); // default value
    Session session = new Session(sessionOptions);
    if (!session.start()) {
        System.out.println("Could not start session.");
        System.exit(1);
    }
    if (!session.openService("//blp/refdata")) {
        System.out.println("Could not open service " +
                          "/blp/refdata");
        System.exit(1);
    }
}
```
The major steps are:

- A Session is created and started; then that Session is used to open a service named "/blp/refdata", a service that provides data according to the Request/Response paradigm.

  In this example, the values explicitly set for host and port correspond to the default values for Session; supply the values for your installation. If the default values suffice then Session construction can be simplified to:

  ```java
  Session session = new Session();
  ```

- The Session is used to obtain refDataSvc, a handle for the service, which is used to obtain an empty Request object for the "ReferenceDataRequest" operation.

- The empty request object is customized to the data needed for this application: the security of interest is "IBM US Equity", the Bloomberg field of interest is "PX_LAST" (last price).

- The request is sent to the service along with requestID, an application specified CorrelationID. (The value chosen is not important for this example.)

- The application enters a loop that makes a blocking request for nextEvent from the Session. Each Event is handled according to its type.

  - Both PARTIAL_RESPONSE and (final) RESPONSE events are handled by the user defined handleResponseEvent method. The only difference is that
the (final) RESPONSE changes the state of continueToLoop so that the looping stops and the application terminates.

- Event objects of any other type are handled by a different user defined handler, handleOtherEvent.

In this application, the event handlers simply output some information about the received events.

```java
private static void handleResponseEvent(Event event) throws Exception
{
    System.out.println("EventType =" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=" +
                            message.correlationID());
        System.out.println("messageType =" +
                            message.messageType());
        message.print(System.out);
    }
}
```

This handler outputs the key features of the received Event.

- Each Event has a type and possibly some associated Messages which can be obtained via the MessageIterator obtained from the Event.
- Each Message from these response events shows the same CorrelationID that was specified when the Request was sent. Additionally, each Message has a type.
- Finally, there is a print method to output the details of the Message in a default format.

Sample output is shown below:

```java
EventType =RESPONSE
correlationID=User: 1
messageType  =ReferenceDataResponse
ReferenceDataResponse (choice) = {
    securityData[] = {
        securityData = {
            security = IBM US Equity
            sequenceNumber = 0
            fieldData = {
                PX_LAST = 82.14
            }
        }
    }
}
```
However, this response to our query is not the only output from this program. This application also receives Events of type neither PARTIAL_RESPONSE nor RESPONSE.

This output comes from the event handling function called from the default case of the switch statement. The events reported here are returned in response to the applications starting of a session and opening of a service.

```java
private static void handleOtherEvent(Event event) throws Exception {
    System.out.println("EventType=" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=", message.correlationID());
        System.out.println("messageType=" + message.messageType());
        message.print(System.out);
        if (Event.EventType.Constants.SESSION_STATUS ==
            event.eventType().intValue()
            && "SessionTerminated" ==
            message.messageType().toString()){
            System.out.println("Terminating: " +
                message.messageType());
            System.exit(1);
        }
    }
}
```

The overall organization of handleOtherEvent is quite similar to that of handleResponseEvent but there are some notable differences:

- Some messages (e.g., system messages) may not have a CorrelationID. The handler must be able to handle such cases.

  **Note:** The SERVICE_STATUS correlation ID has type Internal because it was automatically generated. The RESPONSE correlation ID that was explicitly specified by the application is typed User.

- There may be events that do not arise from application request; for example, an unexpected session shutdown.
2.4 Using the Subscription Paradigm

Our example application requesting subscription data is quite similar to that shown to illustrate the request/response paradigm. The key differences are shown in bold font.

public static void main(String[] args) throws Exception {
    Create and start session.
    if (!session.openService("//blp/mktdata")) {
        System.err.println("Could not start session.");
        System.exit(1);
    }

    CorrelationID subscriptionID = new CorrelationID(2);
    SubscriptionList subscriptions = new SubscriptionList();
    subscriptions.add(new Subscription("AAPL US Equity",
                                        "LAST_PRICE",
                                        subscriptionID));

    session.subscribe(subscriptions);
    int updateCount = 0;
    while (true) {
        Event event = session.nextEvent();
        switch (event.eventType().intValue()) {
            case Event.EventType.Constants.SUBSCRIPTION_DATA:
                handleDataEvent(event, updateCount++);
                break;
            default:
                handleOtherEvent(event);
                break;
        }
    }
}

- The service opened by this application has been changed from "/blp/refdata" (reference data) a service that follows the request/response paradigm to "/blp/mktdata" (market data), a service that follows the subscription paradigm.

- Instead of creating and initializing a Request; here we create and initialize a SubscriptionList and then subscribe to the contents of that list. In this first example, we subscribe to only one security, "AAPL US Equity", and specify only one Bloomberg field of interest, LAST_PRICE (the subscription analog for PX_LAST, the field used in the request/response example).

- The request/response example had application logic to detect the final event of the request and then break out of the event-wait-loop. Here, there is no final event. A subscription will continue to send update events until cancelled (not done in this example) or until the session shut down (handled, as we did before, in the handleOtherEvent method).

- The event type of particular interest is now SUBSCRIPTION_DATA. In this example, these events are passed to the handleEventData method.
The `handleDataEvent` method is quite similar to `handleResponseMethod`. The additional parameter, `updateCount`, is used in this simple example just to enhance the output.

```java
private static void handleDataEvent(Event event, int updateCount)
    throws Exception {

    System.out.println("EventType=" + event.eventType());
    System.out.println("updateCount = " + updateCount);
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID = " +
            message.correlationID());
        System.out.println("messageType   = " +
            message.messageType());
        message.print(System.out);
    }
}
```

Despite these many similarities, the output from the subscription is considerably different from that of the request/response. Examine the output for a random event in the sequence:

```
EventType=SUBSCRIPTION_DATA
updateCount = 54
correlationID = User: 2
messageType   = MarketDataEvents
MarketDataEvents = {
    LAST_PRICE = 85.71
    VOLUME = 18969874
    LAST_TRADE = 85.71
    LAST_ALL_SESSIONS = 85.71
    EQY_TURNOVER_REALTIME = 1.6440605281984758E9
    ALL_PRICE_SIZE = 100
    ALL_PRICE = 85.71
    SIZE_LAST_TRADE_TDY = 100
    RT_PX_CHG_NET_ID = -4.29
    RT_PX_CHG_PCT_ID = -4.767
    VOLUME_TDY = 18969874
    LAST_PRICE_TDY = 85.71
    LAST2_PRICE = 85.719
    LAST_DIR = -1
    LAST2_DIR = 1
    SIZE_LAST_TRADE = 100
    TIME = 19:06:30.000+00:00
    TRADE_SIZE_ALL_SESSIONS_RT = 100
    EVENT_TIME = 19:06:30.000+00:00
    EID = 14005
    IS_DELAYED_STREAM = false
}
Clearly, this subscription event provides much data in addition to LAST_PRICE, the specifically requested field (shown in bold above). A later example will demonstrate how a customer application can extract and use the value of interest.

**Note:** The Bloomberg infrastructure is at liberty to package additional fields in the data returned to a client; however, the client cannot validly expect any data except the requested fields. This sample output shows that the requested field is the first data out of message; that is happenstance and cannot be assumed.

The output of the `otherEventHandler` method also shows differences from the first example.

```plaintext
EventType=SESSION_STATUS
  correlationID=null
  messageType=SessionStarted
  SessionStarted = {
  }

EventType=SERVICE_STATUS
  correlationID=Internal: 1
  messageType=ServiceOpened
  ServiceOpened = {
  }

EventType=SUBSCRIPTION_STATUS
  correlationID=User: 2
  messageType=SubscriptionStarted
  SubscriptionStarted = {
  }
```

In addition to the events for the start of session and opening of a service, which were seen in the request/response example, we also see here an event signaling that a subscription has been initiated. The empty `SubscriptionStarted` message indicates successful starting of the subscription; otherwise, there would have been error information. The value of the `CorrelationID` informs the customer application which subscription (of possibly many subscription requests) has been successfully started.
3 Sessions and Services

3.1 Sessions

The Session object provides the context of a customer application's connection to the Bloomberg infrastructure via the Bloomberg API. Having a Session object, customer applications can use them to create Service objects for using specific Bloomberg services. Depending on the service, a client can send Request objects or start a subscription. In both cases, the Bloomberg infrastructure responds by sending Event objects to the customer application.

3.2 Services

All Bloomberg data provided by the Bloomberg API is accessed through a "service" which provides a schema to define the format of requests to the service and the events returned from that service. The customer application's interface to a Bloomberg service is a Service object.

Accessing a Service is a two step process.

- Open the Service using either the openService or the openServiceAsync methods of the Session object.
- Obtain the Service object using the getService method of the Session object.

In both stages above, the service is identified by its "name", an ASCII string formatted as "/namespace/service"; for example, "/blp/refdata".

Once a service has been successfully opened, it remains available for the lifetime of that Session object.

3.3 Event Handling

The Bloomberg API is fundamentally asynchronous - applications initiate operations and subsequently receive Event objects to notify them of the results; however, for developer convenience, the Session class also provides synchronous versions of some operations. The start, stop, and openService methods seen in earlier examples encapsulate the waiting for the events and make the operations appear synchronous.

The Session class also provides two ways of handling events. The simpler of the two is to call the nextEvent method to obtain the next available Event object. This method will block until an Event becomes available and is well-suited for single threaded customer applications.
Alternatively, one can supply an `EventHandler` object when creating a `Session`. In this case, the user-defined `processEvent` method in the supplied `EventHandler` will be called by the Bloomberg API when an `Event` is available. The signature for `processEvent` method is:

```java
public void processEvent(Event event, Session session) {
    // Note: no exceptions are thrown
}
```

The calls to the `processEvent` method will be executed by a thread owned by the Bloomberg API, thereby making the customer application multi-threaded; consequently customer applications must, in this case, ensure that data structures and code accessed from both its main thread and from the thread running the `EventHandler` object are thread-safe.

The two choices for event handling are mutually exclusive:

- If a `Session` is provided with an `EventHandler` when it is created calling the `nextEvent` method will throw an exception.
- If no `EventHandler` is provided then the only way to retrieve `Event` object is by calling the `nextEvent` method.
3.3.1 Synchronous Event Handling

The following code fragments use synchronous methods on the `Session` and single threaded event handling using the `nextEvent` method.

```java
public static void main(String[] args) throws Exception {
    SessionOptions sessionOptions = new SessionOptions();
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);
    Session session = new Session(sessionOptions);
    if (!session.start()) {
        System.out.println("Could not start session.");
        System.exit(1);
    }
    if (!session.openService("//blp/refdata")) {
        System.out.println("Could not open service "+
                        "/blp/refdata");
        System.exit(1);
    }
    Construct a request
    Send the request via session.
    boolean continueToLoop = true;
    while (continueToLoop) {
        Event event = session.nextEvent();
        switch (event.eventType().intValue()) {
            case Event.EventType.Constants.PARTIAL_RESPONSE:
                Handle Partial Response
                break;
            case Event.EventType.Constants.RESPONSE: // final event
                Handle Final Event
                continueToLoop = false;
                break;
            default:
                Handle Other Events
                break;
        }
    }
    session.stop();
    System.exit(0);
}
```
3.3.2 Asynchronous Event Handling

Use of asynchronous event handling shifts many programmatic details from the `main` function to the event handler.

```java
public static void main(String[] args) throws Exception {
    SessionOptions sessionOptions = new SessionOptions();
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);
    Session session = new Session(sessionOptions,
        new MyEventHandler());
    session.startAsync();
    // Wait for events
    Object object = new Object();
    synchronized (object) {
        object.wait();
    }
}
```

The status for starting the asynchronous session will be received as an event and checked in the handler. Also, there is no exit from `main`; logic in the event handler will determine when the process should be terminated.

The `MyEventHandler` class is in this example a non-public class (it is used only by `main`) implementing the `EventHandler` interface. The class also defines `dumpEvent`, a "helper" function.

```java
class MyEventHandler implements EventHandler {
    void dumpEvent(Event event){
        Output event type.
        For each message, output the type and correlation ID.
    }

    public void processEvent(Event event, Session session) {
        Details below.
    }
}
```
The `processEvent` method is organized to each of the expected events as well as unexpected events:

```java
public void processEvent(Event event, Session session) {
    switch (event.eventType().intValue()) {
        case Event.EventType.Constants.SESSION_STATUS: {
            // If session started, open service.
            break;
        }
        case Event.EventType.Constants.SERVICE_STATUS: {
            // If service opened successfully, send request.
            break;
        }
        case Event.EventType.Constants.PARTIAL_RESPONSE: {
            // Handle partial response.
            break;
        }
        case Event.EventType.Constants.RESPONSE: {
            // Handle final response.
            break;
        }
        default: {
            // Handle unexpected response.
            break;
        }
    }
}
```

Each case in `processEvent` will now be examined in greater detail.

We first show the processing of the event returned for starting the session. If successful, the code will attempt to open the needed service. Since the `openServiceAsync` method throws an exception on failure, but `processEvent` is not allowed to emit an exception, that call must be surrounded by a `try-catch` block. In event of failure, this simple example chooses to terminate the process.
On receipt of a SERVICE_STATUS type event, the messages are searched for one indicating that the openServiceAsync call was successful: the message type must be "ServiceOpened" and the correlation ID must match the value assigned when the request was sent.
If the service was successfully opened, we can create, initialize and send a request as has been shown in earlier examples. The only difference is that the call to `sendRequest` must be guarded against the transmission of exceptions, not a concern until now.

```java
case Event.EventType.Constants.SERVICE_STATUS: {
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        if (message.correlationID().value() == 99
            && message.messageType().equals("ServiceOpened")) {
            // Construct and issue a Request
            Service service = session.getService("//blp/refdata");
            Request request =
                service.createRequest("ReferenceDataRequest");
            request.append("securities", "IBM US Equity");
            request.append("fields", "LAST_PRICE");
            try {
                session.sendRequest(request, new CorrelationID(86));
            } catch (Exception e) {
                System.err.println("Could not send request");
                System.exit(1);
            }
        } else {
            // Handle other message types, if expected.
        }
    }
    break;
}
```

The handling of events containing the requested data is quite similar to the examples already seen. One difference is that, in this example, on the final event, we terminate the process from the event handler, not from `main`. 
Finally, for completeness, there is a default case to handle events of unexpected types.

default: {
    System.err.println("unexpected Event");
    dumpEvent(event);
    System.exit(1);
    break;
}

3.4 Multiple Sessions

Most applications will only use a single Session; however, the Bloomberg API allows the creation of multiple Session objects. Multiple instances of the Session class contend for nothing and thus allow for efficient multi-threading.

For example, a customer application can increase its robustness by using multiple Session objects to connect to different instances of the Server API process.

For another example, a customer application may need from a service both large, heavyweight messages that require much processing as well as small messages that can be quickly processed. If both were obtained through the same session, then the processing of the heavy messages would increase latency on the lightweight messages. That situation can be mitigated by handling the two categories of data with different Session objects and different threads.
4 Requests and Responses

The examples in earlier chapters have shown how to send requests for data and how to handle the corresponding responses. This chapter examines in greater depth the techniques for composing those requests and for extracting data from the response.

The example to be used here, a variation on those already covered, has the same overall organization.

```
import classes
public class RequestResponseExample {
    private static void handleResponseEvent(Event event) throws Exception {
        ..........
    }
    private static void handleOtherEvent(Event event) throws Exception {
        ..........
    }
    public static void main(String[] args) throws Exception {
        create session; start session; open service
        create and initialize request
        send request
        loop until final response is received
    }
}
```

Our focus will be on the creation and initialization of the request in `main` and, later, on the extraction of data from the response in the user-defined `handleResponseEvent` method.

4.1 The Programming Example

The example explored in this chapter is `RequestResponseMultiple.java`. A complete listing of this example and its output can be found in “Request Response Multiple” on page 239.

Translations of `RequestResponseMultiple.java` to the other supported programming languages are also provided:

- `RequestResponseMultiple.cs` (“Request Response Multiple” on page 274)
- `RequestResponseMultiple.cpp` (“Request Response Multiple” on page 299)
- `RequestResponseMultiple.c` (“Request Response Multiple” on page 332)
4.2 Elements

The services provided by the Bloomberg API collectively accept a great variety of different types of requests which, in turn, often take many different parameters and options. The data returned in response is correspondingly diverse in type and organization. Consequently, requests and responses are composed of Element objects: instances of a class with great flexibility in representing data.

- Firstly, an Element object can contain a single instance of a primitive type such as an integer or a string. Secondly, Element objects can also be combined into hierarchical types by the mechanism of SEQUENCE or CHOICE.
  - A SEQUENCE is an Element object that contains one or more Element objects, each of which may be of any type, similar to a struct in the C language.
  - A CHOICE is an Element object that contains exactly one Element object of a type from a list of possible Element types. That list can be composed of any Element types, similar to a union in the C language.
  - Element objects of the SEQUENCE and CHOICE categories can be nested to arbitrary levels.

- Finally, every Element is capable of representing an array of instances of its type.

The Element class also provides introspective methods (in addition to the introspective methods provided by the Java language) which allow the programmatic discovery of the structure of an Element object and any constituent Element objects. However, that level of generality is required in few applications. Most applications can be written to a known structure for request and response, as defined in the schema for a service. Should an application’s structural assumptions prove incorrect (e.g., service schemas can be redefined), then an Exception is generated at run-time.

Note: Incompatible changes to the schema of a Bloomberg core service are very rare. In fact, so far there have been none. Should such changes ever be necessary, they will be phased in and announced with ample warning.

4.3 Request Details

An earlier example showed how to request a single data item (a Bloomberg “field”) for a single security from the Reference Data Service. However, the Reference Data Service accepts more general requests. The service specifies that each "ReferenceDataRequest" can contain three Element objects:

- a list of fields of interest, each a string type,
- a list of securities of interest, each a string type, and
- a list of overrides, each of type FieldOverride, a non-primitive type. This last Element is optional and will not be used in this example.

Our present example begins much as before:
the Session is created and started
the Service is opened and a handle to that Service is obtained.

These steps are performed by the following code fragment:

```java
Session session = new Session();
session.start();
session.openService("//blp/refdata");
Service refDataSvc = session.getService("//blp/refdata");
```

Given the handle to the service, here named refDataSvc, a Request can be created for the request type named "ReferenceDataRequest".

```java
Request request = refDataSvc.createRequest("ReferenceDataRequest");
```

As described in the schema, this request consists of three Element objects named "securities", "fields", and "overrides", each initially empty. These elements represent arrays of strings so their values can be set by appending strings to them specifying the securities and fields required, respectively.

```java
request.getElement("securities").appendValue("AAPL US Equity");
request.getElement("securities").appendValue("IBM US Equity");
request.getElement("securities").appendValue("BLAHBLAH US Equity");
request.getElement("fields").appendValue("PX_LAST"); // Last Price
request.getElement("fields").appendValue("DS002");   // Description
request.getElement("fields").appendValue("VWAP_VOLUME");
// Volume used to calculate the Volume Weighted Average Price (VWAP)
```

The request is now ready to be sent. Note that one of the securities was deliberately set to an invalid value; later, we will examine the error returned for that item.

**Note:** This usage pattern of appending values of arrays of Elements occurs so frequently that the Request class provides convenience methods that are more concise (but also obscure the Element sub-structure):

```java
request.append("securities", "AAPL US Equity");
request.append("securities", "IBM US Equity");
request.append("securities", "BLAHBLAH US Equity");
request.append("fields", "PX_LAST");
request.append("fields", "DS002");
request.append("fields", "VWAP_VOLUME");
```
The rest of `main`, specifically the event-loop for the response, is essentially the same as that used in earlier examples. The `main` function is shown in its entirety below;

```java
public static void main(String[] args) throws Exception {
    Session session = new Session();
    session.start();
    session.openService("//blp/refdata");
    Service refDataSvc = session.getService("//blp/refdata");

    Request request = refDataSvc.createRequest("ReferenceDataRequest");
    request.getElement("securities").appendValue("AAPL US Equity");
    request.getElement("securities").appendValue("IBM US Equity");
    request.getElement("securities").appendValue("BLAHBLAH US Equity");
    request.getElement("fields").appendValue("PX_LAST"); // Last Price
    request.getElement("fields").appendValue("DS002"); // Description
    request.getElement("fields").appendValue("VWAP_VOLUME"); // Volume used to calculate Volume Weighted Average Price (VWAP)

    session.sendRequest(request, new CorrelationID(1));
    boolean continueToLoop = true;
    while (continueToLoop) {
        Event event = session.nextEvent();
        switch (event.eventType().intValue()) {
            case Event.EventType.Constants.RESPONSE: // final response
                continueToLoop = false; // fall through
            case Event.EventType.Constants.PARTIAL_RESPONSE:
                handleResponseEvent(event);
                break;
            default:
                handleOtherEvent(event);
                break;
        }
    }
}
```

### 4.4 Response Details

The response to a "ReferenceDataRequest" request is an element named "ReferenceDataResponse", an `Element` object which is a `CHOICE` of an `Element` named "responseError" (sent, for example, if the request was completely invalid or if the service is down) or an array of `Element` object named "securityData", each containing some requested data. The structure of these responses can be obtained from the service.
schema, but is also conveniently viewed, as we have done earlier, by printing the response in the response event handler code.

```json
ReferenceDataResponse (choice) = {
  securityData[] = {
    securityData = {
      security = AAPL US Equity
      sequenceNumber = 0
      fieldData = {
        PX_LAST = 173.025
        DS002 = APPLE INC
        VWAP_VOLUME = 3.0033325E7
      }
    }
  }
}
```

The fact that the element named "ReferenceDataResponse" is an array allows each response event to receive data for several of the requested securities. The Bloomberg API may return a series of Message objects (each containing a separate "ReferenceDataResponse") within a series of Event objects in response to a request. However, each security requested will appear in only one array entry in only one Message object.

Each element of the "securityData" array is a SEQUENCE that is also named "securityData". Each "securityData" SEQUENCE contains an assortment of data including values for the fields specified in the request. The reply corresponding to the invalidly named security, "BLAHBLAH US Equity", shows that the number and types of fields in a response can vary between entries.

```json
ReferenceDataResponse (choice) = {
  securityData[] = {
    securityData = {
      security = BLAHBLAH US Equity
      securityError = {
        source = 100::bbdbs1
        code = 15
        category = BAD_SEC
        message = Unknown/Invalid security [nid:100]
        subcategory = INVALID_SECURITY
      }
      sequenceNumber = 2
      fieldData = {
      }
    }
  }
}
```

This response message has an Element not previously seen, named "securityError". This Element provides details to explain why data could not be provided for this security. Note that sending one unknown security did not invalidate the entire request.
Just printing the response in the default format is educational but to perform any real work with the response the values must be extracted from the received message and assigned elsewhere for use. The following event handler shows how to navigate the Element structure of the "ReferenceDataResponse".

The `asElement` method of `Message` provides a handle for navigating the contents of the `Message` objects using `Element` methods. If an `Element` object is an array (e.g., `securityDataArray`) then the `numValues` method provides the number of items in the array.

**Note:** The `Element` class also provides similarly named method, `numElements` (not used in this example), which returns the number of `Element` objects in a `SEQUENCE`. 
When stepping through the `securityData` array, the requested Bloomberg fields are accessed by the name and type (e.g., `getElementAsFloat64`, `getElementAsInt32`) as specified in the schema. Once values have been assigned to
local variables they can be used as needed. In this simple example, they are merely output individually in a distinctive format. The program output is shown below.

```plaintext
* security      =AAPL US Equity
* sequenceNumber=0
* px_last       =173.025
* ds002         =APPLE INC
* vwap_volume   =3.0033325E7

* security      =IBM US Equity
* sequenceNumber=1
* px_last       =126.46
* ds002         =INTL BUSINESS MACHINES CORP
* vwap_volume   =2885962.0

* security      =BLAHBLAH US Equity
securityError = {
    source = 100::bbdbs1
    code = 15
    category = BAD_SEC
    message = Unknown/Invalid security [nid:100]
    subcategory = INVALID_SECURITY
}
```

The `sequenceNumber` is provided to allow the ordering of `PARTIAL_RESPONSE` events from the reference data service.
5 Subscriptions

Subscriptions are ideal for data that changes frequently and/or at unpredictable intervals. Instead of repeatedly polling for the current value your application gets the latest value as soon as it is available without wasting time and bandwidth when there has been no change.

This chapter contains more details on how you can start, modify, and stop subscriptions as well as what to expect as the result of a subscription and how to handle those results. This chapter uses examples from the "/blp/mktdata" service.

Currently, the Bloomberg API services that provide a subscription service are market data and Custom VWAP. In the future, the Bloomberg API may support delivering information other than market data through a subscription service.

5.1 Starting a Subscription

There are four parts to creating a subscription; however several have default values:

- The **service name** (for example, "/blp/mktdata"). If you do not specify the service name the defaultSubscriptionService of the SessionOptions object is used.

- The topic. In the case of "/blp/mktdata" the topic value consists of an optional symbology identifier followed by an instrument identifier. For example, "/cusip/097023105" and "/sedol1/2108601" include the symbology identifier whereas "IBM US Equity" omits the symbology identifier. If you do not specify the symbology identifier then the defaultTopicPrefix of the SessionOptions object is used.

  **Note:** The topic's form may be different for different subscription services.

- The **options**. These are qualifiers that can affect the content delivered. Examples in "/blp/mktdata" include specifying which fields an application requires or specifying an interval for conflated data.

- The **correlation ID**. Data for each subscription is tagged with a correlation ID (represented as a CorrelationID object) which must be unique to the session. The customer application can specify that value when the subscription is created. If the customer application does not specify a correlation ID, the Bloomberg infrastructure will supply a suitable value; however, in practice, the internally generated correlation ID is rarely used. Most customer applications assign meaningful correlation ids that allow the mapping of incoming data to the originating request or subscription.

You can represent any subscription as a single string that includes the service name, topic and options. For example:

- "/blp/mktdata/cusip/097023105?fields=LAST_PRICE,LAST_TRADE_ACTUAL" represents a subscription using the market data service to an instrument (BA) specified by CUSIP
where any changes to the fields LAST_PRICE or LAST_TRADE_ACTUAL from the Bloomberg data model should generate an update.

- "IBM US Equity?fields=BID,ASK&interval=2" represents a subscription using the market data service to an instrument (IBM) specified by Bloomberg Ticker where any changes to the fields BID or ASK from the Bloomberg data model should generate an update subject to conflation restriction of at least two seconds between updates. In this case, we are assuming that the Session has a defaultSubscriptionService of "/blp/mktdata" and a defaultTopicPrefix of "ticker/".

The Bloomberg API provides methods which accept the subscription specification as a single string as well as methods in which the different elements of the subscription are specified as separate parameters. Subscriptions are typically manipulated in groups so the Bloomberg API provides methods that operate on a list of subscriptions. This example shows subscription creation by several of these methods.

```
SubscriptionList subscriptions = new SubscriptionList();
CorrelationID subscriptionID_IBM = new CorrelationId(10);
subscriptions.add(new Subscription("IBM US Equity", "LAST_TRADE", subscriptionID_IBM));
subscriptions.add(new Subscription("/ticker/GOOG US Equity", "BID,ASK,LAST_PRICE", new CorrelationID(20)));
subscriptions.add(new Subscription("MSFT US Equity", "LAST_PRICE", "interval=.5", new CorrelationID(30)));
subscriptions.add(new Subscription("/cusip/097023105?fields=LAST_PRICE&interval=5.0", //BA US Equity new CorrelationID(40)));
session.subscribe(subscriptions);
```

**NOTE:** SubscriptionList in C# is simply an alias to System.Collections.Generic.List<Bloomberglp.Blpapi.Subscription>, created with:

```
SubscriptionList sl = new SubscriptionList();
sl.Add(new Subscription("4444 US Equity"));
```

Subscribing to this list of subscriptions returns an Event of type SUBSCRIPTION_STATUS consisting of a Message object of type SubscriptionStarted for each
CorrelationID. For example, the user-defined "dump" method used previous examples shows:

```plaintext
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionStarted
CorrelationID=User: 10
SubscriptionStarted = {
}
messageType=SubscriptionStarted
CorrelationID=User: 20
SubscriptionStarted = {
}
messageType=SubscriptionStarted
CorrelationID=User: 30
SubscriptionStarted = {
}
messageType=SubscriptionStarted
CorrelationID=User: 40
SubscriptionStarted = {
}
```

In case of an error, there is an Event to report the subscriptions that failed. For example, if the specification for MSFT (correlation ID 30) above was mistyped (MSFTT) we would get the event:

```plaintext
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionFailure
CorrelationID=User: 30
SubscriptionFailure = {
    reason = {
        source = BBDB@p111
        errorCode = 2
        category = BAD_SEC
        description = Invalid security
    }
}
```

5.2 Receiving Data from a Subscription

Once a subscription has started, the application will receive updates for the requested data in Message objects arriving Event objects of type SUBSCRIPTION_DATA. With each message there is a CorrelationID to identify the subscription that requested the data.

The "/blp/mktdata" service typically responds with Message's which have more data than was requested for the subscription. In our example, only updates to the LAST_TRADE field of IBM were requested in the subscription corresponding to CorrelationID 10. Applications must be prepared to extract the data they need and to discard the rest.
5.3 Modifying an Existing Subscription

Once you have created a subscription you may modify the options (for example, to change the fields you wish to receive) using the resubscribe method of Session.

**Note:** Use of the resubscribe method is generally preferred to cancelling the subscription (using the unsubscribe method) and creating a new subscription because updates might be missed between the unsubscribe and subscribe calls.

As we saw with the subscribe method, the resubscribe method takes a SubscriptionList. For example, to change the fields reported in the subscription

```python
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 10
MarketDataEvents = {
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
    IS_DELAYED_STREAM = true
    TIME = 14:34:44.000+00:00
    VOLUME = 7589155
    RT_OPEN_INTEREST = 8339549
    RT_PX_CHG_PCT_1D = -0.32
    VOLUME_TDY = 7589155
    LAST_PRICE = 118.15
    HIGH = 118.7
    LOW = 116.6
    LAST_TRADE = 118.15
    OPEN = 117.5
    PREV_SES_LAST_PRICE = 118.53
    EQY_TURNTOVER_REALTIME = 8.93027456E8
    RT_PX_CHG_NET_1D = -0.379999
    OPEN_TDY = 117.5
    LAST_PRICE_TDY = 118.15
    HIGH_TDY = 118.7
    LOW_TDY = 116.6
    RT_API_MACHINE = p240
    API_MACHINE = p240
    RT_PRICING_SOURCE = US
    EXCH_CODE_LAST = D
    EXCH_CODE_BID = O
    SES_START = 09:30:00.000+00:00
    SES_END = 16:30:00.000+00:00
}
```
created earlier with the correlation ID of `subscriptionID_IBM` we can use the following code fragment:

```java
SubscriptionList subscriptions = new SubscriptionList();
subscriptions.add(new Subscription("IBM US Equity",
    "BID,ASK",
    subscriptionID_IBM));
session.resubscribe(subscriptions);
```

The client receives an `Event` object indicating successful re-subscription (or not) before receipt of any data from that subscription.

**Note:** The behavior is undefined if the topic of the subscription (e.g., the security itself) is changed.

### 5.4 Stopping a Subscription

The Bloomberg API provides an `unsubscribe` method that will cancel a single subscription (specified by its `CorrelationID`) and another method that will cancel a list of subscriptions. The following code fragment cancels all of the subscriptions created earlier.

```java
SubscriptionList subscriptions = new SubscriptionList();
for (int id = 10; id <= 40; id += 10) {
    subscriptions.add(new Subscription("IBM US Equity",
        new CorrelationID(id)));
    // Note: The topic string is ignored for unsubscribe.
}
session.unsubscribe(subscriptions);
```

**Note:** No `Event` is generated for `unsubscribe`.

### 5.5 Overlapping Subscriptions

Your application may make subscriptions that “overlap”.

One form of overlap occurs when a single incoming update may be relevant to more than one subscription. For example, two or more subscriptions may specify the updates for the same data item. This can easily happen inadvertently by "topic aliasing": one subscription specifies a security by ticker, the other by CUSIP.

Another form of overlap occurs when separate data items intended for different subscriptions on the customer application process arrive in the same `Message` object.
For example, the Bloomberg infrastructure is at liberty to improve performance by packaging two data items within the same Message object. This can occur when a customer’s application process has made two separate subscriptions, where one includes a request for "IBM US Equity" and "LAST_TRADE", while the second one includes "IBM US Equity" and "LAST_TRADE".

The customer application developer can specify how the Bloomberg API should handle overlapping subscriptions. The behavior is controlled by for the allowMultipleCorrelatorsPerMsg option to the SessionOptions object accepted by the Session constructor.

If the allowMultipleCorrelatorsPerMsg option is false (the default) then a Message object that matches more than one subscription will be returned multiple times from the MessageIterator, each time with a single, different CorrelationID.

If the allowMultipleCorrelatorsPerMsg object is true then a Message object that matches more than one subscription will be returned just once from the MessageIterator. The customer application developer must supply logic to examine the multiple correlation ID values (see the numCorrelationIds and correlationIDAt methods of the Message class) and dispatch the appropriate data to the correct application software.

5.6 Conflation and the Interval Option

The API will conflate data only when requested with the Interval option on a subscription. If multiple subscriptions exist for the same security across a range of intervals then the API will have a single subscription from the Bloomberg cloud which is then "intervalized" as appropriate and distributed to individual subscribers.

5.7 Delayed Data

Delayed Data (data for users / applications that are not explicitly entitled to real-time data) is generally pre-conflated before leaving the Bloomberg cloud for client-side applications. Please note that Desktop API and Server API will have automatic access to delayed data (where available), whereas B-Pipe requires explicit permission for access.

5.8 Subscription Life Cycle

There are several key points in the life cycle of a subscription:

- **Start-up**: Subscriptions are started by the subscribe method of Session. An Event object is generated to report the successful creation of any subscriptions and separate events for each failure, if any.
- **Data Delivery**: Data is delivered in Event objects of type SUBSCRIPTION_DATA; each such event has one or more messages; each such Message object has one
or more correlation IDs to identify the associated subscriptions. Since each 
Message object may contain more data than requested in any individual 
subscription, the code managing each subscription must be prepared to extract its 
data of interest from the Message object.

Note: customer applications must not rely on the delivery of data that was not 
explicitly requested in the subscription.

- **Modification:** A list of subscriptions (each subscription identified by its correlation 
  ID) can be modified by the resubscribe method of Session.

- **Cancellation:** Subscriptions (each subscription identified by its correlation ID) can 
  be cancelled by the unsubscribe method of Session.

- **Failure:** A subscription failure (e.g., a server-side failure) is indicated by an Event 
  of type SUBSCRIPTION_STATUS containing a Message to describe the problem.
6 Authorization and Permissioning Systems

6.1 Overview

It is necessary to restrict access to data to users who are entitled to view it. With the Bloomberg API data products this is essentially a three step process.

**Authentication**
Who is the consumer?

**Authorization**
What data is the consumer entitled to see?

**Permissioning**
The process of enforcing data distribution to only entitled consumer.

6.2 Underlying Concepts

6.2.1 EIDs

EIDs are integers that represent the entitlement for a security's source (e.g. a level 1 entitlement for MSFT UQ Equity would have an EID of 14005, level 2 data would be additional EIDs).

Instruments from a common source (e.g., NASDAQ) will share an EID; for example, MSFT UQ Equity and INTC UQ Equity both come from NASDAQ and so have EID 14005 (if requested by someone with level 1 access).

Users and applications can have EIDs associated with them to represent their entitlements. For a BLOOMBERG PROFESSIONAL service user, this is the same as the entitlements on the BLOOMBERG PROFESSIONAL service.

6.2.2 Requirement for the Terminal

The licence for distribution of data to existing BLOOMBERG PROFESSIONAL service users requires that they are logged into the Bloomberg Terminal in order to view the data. In this respect the data products can be seen, for Bloomberg users, as an extension of the Terminal product and thus sharing entitlements and exchange fees with their Terminal account.
Authentication in Bloomberg’s data products for Bloomberg users is performed by identifying a user as being logged into the Terminal. The Terminal’s use of a biometric device will have already proven the identity of the logged in user.

Please note that the Terminal is not a requirement for B-PIPE’s non-BPS (Market Data) users or applications.

6.2.3 The //blp/apiauth service

The authentication and permissioning systems of Server API and B-PIPE require use of the //blp/apiauth service. This defines the requests and responses that will come from the API.

6.2.4 The V3 Identity Object

V3 permissioning, on both Server API and B-PIPE, revolves around the use of a class called the Identity. These objects represent a user (or an application in B-PIPE) and can be used to check that a user is entitled for data, is logged onto a terminal, switches terminals, and can be passed with a request to receive data permissioned just for that user or application.

6.2.5 V3 Permissioning Models

The V3 API provides two permissioning models for developers to follow.

User mode

When user mode permissioning is used, an Identity is passed as a parameter when sending a request. This means that all data returned will be already permissioned for that Identity, but is only for distribution to that particular user or application represented by the Identity.

Content based

When content based permissioning is used, the entitlement identifiers (EIDs) of incoming pieces of data is taken and the data is only distributed to users whose Identity contains the same EIDs as the data.

6.2.6 Authorization Lifetime

Before designing and developing your Server API or B-PIPE application, it is important that you understand the following guidelines concerning the authorization lifetime of a Bloomberg user:

1. An application requires only one Identity object per session per Bloomberg user. This means that your application is not required to authorize the user each time the user makes a request for data.
2. A Bloomberg user’s authorization remains valid until that user logs out from Bloomberg Professional service and logs in from another host. At that time, your application will receive an event of type AUTHORIZATION_STATUS, containing a message of type AuthorizationRevoked.

This is the only time that an Identity must be re-established.

Simply logging out or logging back in from the same host will not invalidate a user’s authorization.

3. User Authorization is needed when the session is destroyed or when the authorization is revoked.

4. If any entitlements change for the user, the existing Identity object is automatically updated by Bloomberg’s infrastructure and SDK.

Failure to follow these guidelines may result in exceeding the maximum concurrently active authorizations limit for a user or application, thereby resulting in further authorizations failing with error code MAX_AUTHORIZATIONS_EXCEEDED.

Identities can be explicitly cancelled by calling session.cancel on the correlation ID of the authorisation request that populated them.

6.3 Server API Authorization

6.3.1 Authorization by IP Address

Authorization by IP address consists of sending to the Bloomberg infrastructure an authorization request containing a user identify (UUID) and the IP address of the host where that user is believed to be using the BLOOMBERG PROFESSIONAL service. If that user indeed has a Bloomberg session at that IP address, the authorization is successful.

When the customer application has a User Mode deployment, the authorization request is submitted by the end-user application.
When the customer application has a Server Mode deployment, the authorization request is submitted by the customer server application using values obtained by the end-user applications by some customer defined protocol.
The above diagram does not show the subordinate customer application that will be receiving the Bloomberg data. That application must report its user’s UUID and IP address to the customer application using the Server API. The customer application developer must define the protocol for transferring that information.

To authorize a UUID/IP address pair, open "//blp/apiauth", the authorization service, and send an authorization request. The following code fragment shows how to create such a request and one method for blocking until receipt of the corresponding response.
<Java>

int uuid = ..........; // Obtain UUID for user of interest.
String ipAddress = ..........; // Obtain IP address for user of interest.

.......... Create and start 'session'. ..........

if (!session.openService("//blp/apiauth")
{
    System.out.println("Could not open service " + "/blp/apiauth");
    System.exit(1);
}
Service apiAuthSvc = session.getService("//blp/apiauth");
Request authorizationRequest = apiAuthSvc.createAuthorizationRequest();
authorizationRequest.set("uuid", uuid);
authorizationRequest.set("ipAddress", ipAddress);

Identity identity = session.createIdentity();
CorrelationID authorizationRequestID = new CorrelationID(10);

session.sendAuthorizationRequest(authorizationRequest, identity,
authorizationRequestID);

System.out.println("sent Authorization Request using ipAddress");

// Wait for 'AuthorizationSuccess' message which indicates 
// that 'identity' can be used.
The “helper” method, handleAuthenticationResponseEvent, examines the received messages for one of type "AuthorizationSuccess", "AuthorizationFailure", etc.

```java
static private boolean handleAuthenticationResponseEvent(Event event) throws IOException {
    if (hasMessageType(event, "AuthorizationSuccess")) {
        System.err.println("Authorization OK");
        return true;
    } else if (hasMessageType(event, "AuthorizationFailure")) {
        System.err.println("Authorization Problem");
        dumpEvent(event);
    } else {
        System.err.println("Authorization: Other Problem");
        dumpEvent(event);
    }
    return false;
}
```
For a valid UUID/IP address pair, the program output is:

```
sent Authorization Request using ipAddress
  EventType=SESSION_STATUS
  correlationID=null
  messageType=SessionStarted
  SessionStarted = {
  }
  EventType=SERVICE_STATUS
  correlationID=Internal: 1
  messageType=ServiceOpened
  ServiceOpened = {
  }
  Authorization OK
```

Successful authorization loads identity with information (i.e., entitlement data) later used in the Permissioning phase.

However, if incorrect data is given, say an incorrect IP address, the output is:

```
sent Authorization Request using ipAddress
  EventType=SESSION_STATUS
  correlationID=null
  messageType=SessionStarted
  SessionStarted = {
  }
  EventType=SERVICE_STATUS
  correlationID=Internal: 1
  messageType=ServiceOpened
  ServiceOpened = {
  }
  Authorization Problem
  eventType=RESPONSE
  messageType=AuthorizationFailure
  CorrelationID=User: 10
  AuthorizationFailure = {
    reason = {
      code = 102
      message = User not logged on to the Bloomberg Professional Service
      category = NO_AUTH
      subcategory = NOT_LOGGED_IN
      source = [nydsmeter1]
    }
  }
  Authorization Failed
```
6.4 B-PIPE Authorization

**Note:** B-PIPE requires an **Identity** to be passed with every subscription and data request; this **Identity** can either be a **User** or an **Application**.

B-PIPE Authorization requires prior administrative action to enable each user and/or application.

Please contact your firm's Bloomberg **EMRS** administrator.

There are two programmatic stages to B-PIPE Authorization:

- "Authentication" of identity. This can be by user and/or by application
- "Authorization" which is the process of obtaining the entitlements of the authenticated user and/or application

B-PIPE authentication and authorization is displayed in Figure 6-3.

![Figure 6-3: Obtaining a User’s Identity in B-PIPE](image-url)
Figure 6-3 shows the procedure for the user authorization system. It is important to note that the "authentication" section of the diagram MUST be performed on the user's desktop machine. The "authorization" section can be performed on the server-side application or on the user's desktop, depending on the application.

For an application authorization system, the OS_LOGIN or DIRECTORY_SERVICE request is replaced with one for the Application Name as defined on EMRS and this can be run on any machine.

For a combined application and user authorization system both the user authentication and the application authentication occurs in a single call and this must be run on the user desktop machine.

6.4.1 Authentication

The first stage of authentication is creating an Authentication Options string. This is attached to the SessionOptions object and thus passed into the session when it is created.

For a User

A user's identity can be authenticated by the user's Window's logon identity or a value from the Active Directory (e.g., email address) associated with the login. The correct authentication value for each user is made known to the Bloomberg Data Center using the EMRS function.

The client application specifies this choice using the setAuthenticationOptions method of the SessionOptions class. Note that neither option requires the user to input or even be aware of the value that is used for authentication.

The two options are OS_LOGON and DIRECTORY_SERVICE.

An example of their use is as follows:

```c
const char *authenticationOptions = "AuthenticationType=OS_LOGON"
const char *authenticationOptions = "AuthenticationType=DIRECTORY_SERVICE;
DirSvcProperty=mail";
```

"mail" is the property name to lookup under Active Directory rather than the value itself. The libraries will obtain the value from Active Directory using this property name for the currently logged in user.

A code example demonstrating the use of these can be found below in Token Generation.

For an Application

An application "authenticates" in much the same way as a user. However, instead of using Active Directory or a Logon, an application name is used as defined in EMRS.

Rather than using OS_LOGON and DIRECTORY_SERVICE with the AuthenticationType parameter of the authentication options string, we introduce two new parameters; AuthenticationMode and ApplicationAuthentication.
**AuthenticationMode** will take the value APPLICATION_ONLY and **ApplicationAuthentication** will take the value APPNAME_AND_KEY.

Finally we use the parameter **ApplicationName**. The value for this parameter will be the value stored on EMRS for that application.

```c
const char *authenticationOptions =  "AuthenticationMode=APPLICATION_ONLY; ApplicationAuthenticationType=APPNAME_AND_KEY; ApplicationName=TestApplication"
```

The above code snippet can be inserted in the following code example to generate a token for an application registered on EMRS as "TestApplication".

After the token is generated, it should then be used to generate an Identity in the same way that a user has an identity created using a token.

There is one last possible value for **AuthenticationMode**: USER_AND_APPLICATION.

This allows use of the **AuthenticationType** parameter with OS_LOGON and DIRECTORY_SERVICE alongside the **AuthenticationMode**, **ApplicationAuthenticationType**, and **ApplicationName** parameters.
const char *authenticationOptions =
"AuthenticationMode=USER_AND_APPLICATION;
ApplicationAuthenticationType=APPNAME_AND_KEY;
ApplicationName=TestApplication;
AuthenticationType=OS_LOGON"

Typically this will be used for authorizing specific users for specific applications and will return the intersection of the entitlements of the application and the user.

6.4.2 Token Generation

The authentication occurs when the client application requests the generation of a "token". A failure to authenticate is indicated by a message of type "TokenGenerationFailure". If a "TokenGenerationSuccess" message is received, the application can extract a token for use in the subsequent Authorization stage. By passing the Authentication Options string in as part of the session options, the call to session.generateToken will submit a token generation request.
// ManagedBpipeAuthorization.cpp
using namespace BloombergLP;
using namespace blpapi;

const char *authenticationOptions
    = useLogon
        ? "AuthenticationType=OS_LOGON"
        : "AuthenticationType=DIRECTORY_SERVICE;DirSvcProperty=mail";

SessionOptions sessionOptions;
sessionOptions.setServerHost("localhost"); //default
sessionOptions.setServerPort(8194); //default

sessionOptions.setAuthenticationOptions(authenticationOptions);

Session session(sessionOptions);

if (!session.start())
{
    std::cerr << "Failed to start session" << std::endl;
    return 1;
}

CorrelationId tokenGenerationId(99);
EventQueue tokenEventQueue;
session.generateToken(tokenGenerationId, &tokenEventQueue);
std::string token;

Event tokenEvent = tokenEventQueue.nextEvent(); // blocking
The token is a long alphanumeric string that has a limited lifespan for validity and needs to be used in an Authorization request before it expires.

### 6.4.3 Identity Object

B-PIPE requires an Identity to be passed with every subscription and data request; this Identity can either be a User or an Application.

Please note that for an application that has been named in EMRS, all requests for data must have the Identity passed with it, so that only the securities that the application is entitled for are accessible rather than everything associated with the B-PIPE.

### 6.5 Authorization

For B-PIPE Authorization, the client application must set as an attribute of the Authorization request the token obtained during Authentication. Then, as in the other cases, an "AuthorizationFailure" message indicates failure (with details) and an "AuthorizationSuccess" message indicates that the identity has been set with the user's or application's entitlements.

The Identity is then used in the same way as it would be in Permissioning in Server API.

Please note that for an application that has been named in EMRS, all requests for data must have the Identity passed with it, so that only the securities that the application is entitled for are accessible rather than everything associated with the B-PIPE.
C++

......authentication stage......

const char *authorizationServicePath = "//blp/apiauth";
if (!session.openService(authorizationServicePath))
{
    std::cerr << "Failed to open "
              << authorizationServicePath
              << std::endl;
    return 1;
}

Service authorizationService =
    session.getService(authorizationServicePath);

Identity identity = session.createIdentity();
Request authorizationRequest =
    authorizationService.createAuthorizationRequest();
authorizationRequest.set("token", token.c_str());

CorrelationId authorizationRequestId(98);

EventQueue authorizationEventQueue;

session.sendAuthorizationRequest(authorizationRequest,
                      &identity,
                      &authorizationRequestId,
                      &authorizationEventQueue);

Event authorizationEvent = authorizationEventQueue.nextEvent();

for (MessageIterator messageIterator(authorizationEvent);
          messageIterator.next(); )
{
    Message message = messageIterator.message();

    if (AUTHORIZATION_FAILURE == message.messageType())
        std::cerr << "Failed authorization" << std::endl;
    return 1;
}

assert(AUTHORIZATION_SUCCESS == message.messageType());
break;
}

......rest of client application......
6.6 Permissioning

6.6.1 Entitlements

Entitlement Identifiers (EIDs) are numeric values associated with data provided by Bloomberg. The following table contains some EID examples:

<table>
<thead>
<tr>
<th>EID</th>
<th>Description</th>
<th>Source</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>14005</td>
<td>NASDAQ Level 1</td>
<td>NASDAQ</td>
<td>MSFT UQ Equity, INTC UQ Equity</td>
</tr>
<tr>
<td></td>
<td>INTC UQ Equity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>BGN</td>
<td>Bloomberg Generic</td>
<td>CT2@BGN Govt</td>
</tr>
<tr>
<td>23599</td>
<td>U.S. Treasuries</td>
<td>Merrill Lynch</td>
<td>CT2@ML Govt</td>
</tr>
<tr>
<td>14014, 14076</td>
<td>London Stock Exchange Level 1 &amp; 2</td>
<td>LSE</td>
<td>VOD LN Equity</td>
</tr>
</tbody>
</table>

a. In the example above, MSFT UQ Equity and INTC UQ Equity are both NASDAQ Level 1, and have the same EID.
b. There can be cases where there are no entitlements associated with the associated instrument. In such cases the data is to be considered free for all BBA users. Bloomberg Generic Pricing has no EID and is therefore, free for all Bloomberg users.
c. In the example above, we show that separate EIDs are used to represent London Stock Exchange Level 1 and Level 2.

The user's EIDs (in the first row, above) are returned in the AuthorizationResponse and are held in an "Identity". Each Message contained in a SUBSCRIPTION_DATA, PARTIAL_RESPONSE or RESPONSE Event may contain an EID field.

Note that for reference data, EIDs are currently assigned at the instrument level, not at the field level. However, for subscription data, EIDs are currently assigned at the instrument and field level.

The following code fragments show how the entitlements loaded into the Identity during the authorization stage and can be used to check a user's eligibility to receive given data.
First, the data request must be modified to request that entitlement identifiers be included with the returned data. For example:

```java
Service refDataSvc = session.getService("//blp/refdata");
Request request = refDataSvc.createRequest("ReferenceDataRequest");
request.append("securities", "VOD LN Equity");
request.append("fields", "PX_LAST");
request.append("fields", "DS002");
request.append("fields", "VWAP_VOLUME");
request.set("returnEids", true); // new
CorrelationID requestID = new CorrelationID(20);
session.sendRequest(request, requestID);
```


Then, the handler for the resulting events can be modified to use the identity acquired during authorization:

```java
private static void handleResponseEvent(Event event, Identity identity)
    throws IOException {
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        Element ReferenceDataResponse = message.asElement();
        if (ReferenceDataResponse.hasElement("responseError")) {
            handle error
        }
        Element securityDataArray = ReferenceDataResponse.getElement("securityData");
        int numItems = securityDataArray.numValues();
        for (int i = 0; i < numItems; ++i) {
            Element securityData = securityDataArray.getValueAsElement(i);
            String security = securityData.getAttribute("security");
            int sequenceNumber = securityData.getAttribute("sequenceNumber");
            if (securityData.hasElement("securityError")) {
                handle error
            }
            ArrayList missingEntitlements = new ArrayList();
            Element neededEntitlements = (securityData.hasElement("eidData") ? securityData : null);
            if (null == neededEntitlements) {
                forward data to the user
            } else if (identity.hasEntitlements(neededEntitlements, message.service(), missingEntitlements)) {
                forward data to the user
            } else {
```
In this example, data is forwarded to a user who has the entitlements for the security, or if the security has no entitlements.

### 6.6.2 User Mode

In User-Mode permissioning, each request or subscription is accompanied by the Identity object, which was obtained when authorizing the user or application. This is the model that must be followed when requesting data as a named Application.

Data received as a result of requests and subscriptions must be carefully segregated by the application both in memory and in any permanent storage to ensure it is only available to the user whose Identity object was used in the request or subscription. Thus, the requirements here are much more complicated than in the earlier models.

Since, in this scenario, a request can be made on behalf of only one user, the User-Mode model may require creation of multiple requests (or subscriptions) that might have been coalesced into a single request (or subscription) under the other models.

Fortunately, the Bloomberg infrastructure improves efficiency by bundling its replies for subscriptions. (Note that this is not done for requests.) Furthermore, although the replies may be bundled, the customer application is (by default) presented with that data presented multiple times, each with a single CorrelationId. If the customer application wishes to handle fewer albeit more complicated responses, the allowMultipleCorrelationsPerMsg option of SessionOptions should be set to true.

One implication of User-Mode permissioning is that there is no way for an application to retrieve data when none of its users are using the BLOOMBERG PROFESSIONAL service.

Whereas, when using Application-Mode / Server-Mode permissioning, it is possible to retrieve data when none of an application's users are logged in.

### 6.6.3 Content Based

In this approach, the customer application retrieves and stores the entitlements of each of its users. The customer application makes requests and subscriptions using the Identity of the Application. All data returned from the Bloomberg infrastructure is requested to be tagged with the Entitlement Identifiers (EIDs) for that data.
For example,

```java
create and open 'session'........
Service refDataSvc = session.getService("//blp/refdata");
Request request = refDataSvc.createRequest("ReferenceDataRequest");
request.append("securities", "VOD LN Equity");
request.append("fields", "PX_LAST");
request.append("fields", "DS002");
request.append("fields", "VWAP_VOLUME");
request.set("returnEids", true);
```

When the response arrives, the customer application must check that EID against the entitlements of a user before actually delivering the data to that user. A user’s entitlements can be checked by using the hasEntitlements method of the Identity object.

```java
Extract 'securityData' from response message........
```

```java
ArrayList missingEntitlements = new ArrayList();
Element neededEntitlements =
    securityData.hasElement("eidData")
    ? securityData.getElement("eidData")
    : null;
if (null == neededEntitlements)
{
    forward data to the user
}
else if (identity.hasEntitlements(neededEntitlements,
    message.service(),
    missingEntitlements))
{
    forward data to the user
}
else
{
    do not forward data to the user
}
```

Of course, using this strategy, some requests may be satisfied and other rejected.
B-Pipe introduced the concepts of Named Applications. These are setup on EMRS and allow an application to be given entitlements and services to consume. Using the Application authentication system described earlier will result in an Identity that represents the Application and can be used in a user mode style to get data based on the EMRS records.

6.7.1 Single-User

Single-User applications are Desktop applications that take a user identity which has been authorized using the USER_AND_APPLICATION authorization mode. This is used in a User Mode style and results are passed directly back to the specific user.

6.7.2 Multi-User

Multi-User applications are typically Client-Server (N-tier, etc.) architectures and can either follow the user mode or content-based permissioning models. User Identities would be again created using the USER_AND_APPLICATION authorization mode (which also checks to see if the user is entitled to use that application according to records on EMRS).

The application could then either send the user identities with separate requests and correlation IDs to get data for individual users, or it can use its own Identity (created just for the application) to request data (the application Identity is the parameter to the request or subscription function). EIDs could be extracted from the returned data and thus can be used in a Server-mode style by distributing to entitled users.

6.7.3 Derived Data / Non-Display

Use of Derived Data and Non-display applications carries a fee. These are essentially applications where users will never see the raw data going into them. The application would simply make requests using its own Identity and the raw incoming data would never be sent to users.

Derived Data applications may pass "resultant data" to users, and the definition of this "resultant data" is clearly defined in the contract.

6.8 V2 Authorization and Permissioning Models

If you have previously worked with prior versions of the API (the pre-V3 C and .NET API) then it is important to note the changes between pre-V3 and V3 style permissioning.

6.8.1 User Mode

Pre-V3 user mode was tied to an application.
In the C API this involved using the `bb_connect_server_user` call which set the entire application as tied to that user. All requests would be processed using that user's entitlements and settings.

.NET used configuration files (or XmlNode objects) with the ServerApiLicense node to determine the credentials of the user on whose behalf the application was to connect. After MarketDataAdapter.Startup() was called, all requests would have been serviced as that user.

V3 avoids the issue of having to dedicate the entire program to a single user and instead allows multiple users in the same application by using Identities as parameters to requests and subscriptions. The same distribution restrictions as pre-V3 still apply; data downloaded on behalf of a single user cannot be distributed to another user.

### 6.8.2 All-or-None

All-or-none permissioning simply compared the set of entitlements of a user against the set of entitlements of the server. If the user had all of the entitlements of the server then that user was permitted to receive any data from the server without further checks.

Pre-V3 provided calls to check this.

The C API used the `bb_get_authorization` function to check this. If any EIDs were returned then that user did not match the Server on those EIDs and thus would have to be denied access to all data from the server application.

The .NET API used the `LicenseManager.GetRestrictions` call. If it returned EIDs then the user had to be denied access to all data.

V3 removes support for all-or-none systems as these are not considered to be flexible enough. In addition problems were caused by entitlements sometimes being applied to users non-homogenously.

### 6.8.3 Content-Based / Per-Product / Per-Security

The pre-V3 implementation of the content-based, originally known as per-product or per-security, permissioning system involved downloading lists of EIDs for each user and for each security. When data was to be passed to users the application developer was responsible for checking that the security's EIDs were a subset of the user's.

In the C API, the EIDs for securities and users were retrieved via the `bb_get_security_entitlements` and `bb_get_user_entitlements` function calls.

In .NET this was performed using the `LicenseManager.GetSecurityEntitlements` and `LicenseManager.GetUserEntitlements` methods.

This is implemented in the V3 system with some minor changes; the logon check and the user entitlements retrieval are now combined into the request to populate an `Identity`. This request currently differs between Server API and B-PIPE and these processes are detailed later in this document.
6.8.4 Validating Logon Status

In the pre-V3 API it was necessary to perform a separate check to see if a user was logged into the terminal on at a specified IP address.

The C API used the `bb_validate_bllg_logon` function and took the user’s UUID, SID, SID Instance, Terminal SID, Terminal SID Instance, and the IP address of the user’s terminal as parameters.

The .NET API worked the same way using the `TerminalMonitor.GetLogonStatus` method.

In V3 this is implemented as part of the authorization process that eventually populates an Identity. In Server API the user’s UUID and IP address of the terminal is passed as part of the authorization request. In B-PIPE, the operating system logon, or Active Directory property, is used to match a user against values stored in the EMRS administrative function on the terminal in order to obtain a Token to pass in instead of the UUID and IP address.
There are two core and five additional services for accessing Bloomberg data. Each API service operates with either the subscription or request/response paradigm through following well-defined schema. The schema defines the request and request options, with detailed information in "Appendix A Schemas". This chapter provides an overview of each of these services.

**Core:**

- Reference Data Service
  
  ```
  //blp/refdata
  ```

- Market Data Service
  
  ```
  //blp/mktdata
  ```

**Additional:**

- Custom VWAP Service
  
  ```
  //blp/mktvwap
  ```

- Market Bar Subscription Service
  
  ```
  //blp/mktbar
  ```

- API Field Information Service
  
  ```
  //blp/apiflds
  ```

- Page Data Service
  
  ```
  //blp/pagedata
  ```

- Technical Analysis Service
  
  ```
  //blp/tasvc
  ```

- API Authorization
  
  ```
  //blp/apiauth
  ```

**Important Notes:**

1. Each Bloomberg data product using the Bloomberg API may vary in the services available and also the entirety of the service available. Please see the specific product overview to determine which services are available.

2. For information on the B-Pipe-only services, please see "B-Pipe Services" on page 117

### 7.1 Common Concepts

#### 7.1.1 Security/Securities

Where a request allows only a single security to be supplied, the field in the schema is named "security" and is a simple string. Where a single request can handle multiple securities the field in the schema is named "securities" and is defined as an array. For example, each `IntradayTickRequest` can only return information on a single security, whereas `ReferenceDataRequest` can return information on many securities.
A security must conform to the following syntax:

\[/[Topic Prefix]/SYMBOLOGY[@Pricing Source][Exchange]\]

Where **[Topic Prefix]** is one of the following:

- ticker
- cusip
- wpk
- isin
- buid
- sedol1
- sedol2
- sicovam
- common
- bsid
- svm
- cins
- cats
- bbgid

The default format for a security is the Bloomberg ticker format, for example, "IBM US Equity". This format consists of:

**SYMBOLOGY** [Exchange] <Yellow Key>

- **SYMBOLOGY** is required and is the ticker name
- [Exchange] is optional and is a two character mnemonic for the exchange where the security is traded. If you do not specify [Exchange] then the default value for the user or for the Server API process will be used.
- <Yellow Key> is the text equivalent of one of the Bloomberg yellow function keys.

<table>
<thead>
<tr>
<th>Govt</th>
<th>Corp</th>
<th>Mtge</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-Mkt</td>
<td>Muni</td>
<td>Pfd</td>
</tr>
<tr>
<td>Equity</td>
<td>Comdty</td>
<td>Index</td>
</tr>
<tr>
<td>Curncy</td>
<td>Client</td>
<td></td>
</tr>
</tbody>
</table>

**Case Sensitivity**

- The API will adjust the yellow key (Equity, Comdty, Index...) to be in the correct format despite the case that is used. An example is that it will adjust "equity" to "Equity".
- The ticker and source are case sensitive and will need to be specified in the correct casing for it to resolve. The only exception is if all characters are specified in lower case in which the API will always change to upper case for both the ticker and source. Hence "vod ln" and "VOD LN" are the same and will both be successful, however "vOD IN" will not resolve.

### 7.1.2 Pricing Source

Bloomberg allows you to specify a provider's pricing for a specific security or for a universe of securities. However, you must have the providing firm’s approval to use their pricing information. If you do not specify a pricing source then the default value for the user of the Server API process is used.
If you wish to specify which pricing source should be used append @ followed by the pricing source to the security, for example, \\
"/cusip/912828GM6@BGN" or "MSFT@ETPX US Equity". Note for securities in the Currency Yellow Key use a space instead of @ to separate the security from the pricing source, for example, "GBPUSD BAAM Currency".

Corporate, Government, and Preferred securities. To find what pricing sources are available for a security, load the security then type PCS<GO> on your Bloomberg. This will also tell you what your preferences for pricing source are for that class of securities. If a pricing is not listed on this screen, then it is not available through the Bloomberg API.

7.1.3 Fields

Some requests (for example, ReferenceDataRequest or HistoricalDataRequest) as well as subscriptions require you to specify which fields from the Bloomberg data model you wish to receive. When using the Reference Data Service you can specify fields using either the field mnemonic or the CALCRT ID. Returned values have the same name (field mnemonic or CALCRT ID) specified in the request. However, when creating subscriptions you will only receive the mnemonic, even if you are passing the CALCRT ID. Therefore, you will want to use the mnemonic for subscriptions.

You can retrieve information about available fields programmatically using the Bloomberg API Field Information Service ("//blp/apiflds") or you can use FLDS<GO> on your BLOOMBERG PROFESSIONAL service.

7.1.4 Overrides

You can use overrides to change the basis on which Bloomberg calculates a derived field. You can use this facility to perform "what if?" analysis. For example, override the bid price of a bond (PX_BID) and request the bid yield to maturity (YLD_YTM_BID) based on the value you supplied for the bid price.

You can retrieve information about which fields react when a particular field is overridden programmatically by using the Bloomberg API Field Information Service, "/blp/apiflds", or you can use FLDS<GO> on your BLOOMBERG PROFESSIONAL service.

You can specify up to 100 overrides in a single request. The overrides are specified in the request as an array of name/value pairs.

The value you supply is always represented as a string. If the override field requires:

- A date, then the format is <YYYY><MM><DD>, where <YYYY> is a 4-digit year, <MM> is a 2-digit month and <DD> is a 2-digit day. Therefore, August 4, 2010 would be specified as 20100804.
- A decimal value, then you must always use a "." (period) character as the decimal separator regardless of any preferences you may have set in your operating system.
7.1.5 Relative Dates

The start and end date of a HistoricalDataRequest are specified using relative dates. These are represented in a string format and allow a great deal of flexibility.

**Syntax**

The syntax of the Relative Date is:

\[ [A] [+/-nCU] \]

where \([A]\) is the Anchor Date (details below) and \([+/-nCU]\) is the Offset from the Anchor Date (details below). Both parts are optional and the date is the result of applying the specified Offset to the specified Anchor.

- If the Anchor Date is omitted then the current date is used.
- If the Offset is omitted then no offset is applied to the Anchor.
  - An empty string is equal to the current date

In the Offset, \(+/-\) defines the direction of the offset, \(n\) is a non-negative integer multiplier, \(C\) is a Calendar Type, and \(U\) is a Period Unit. The integer multiplier in the Offset is optional

**Anchor**

You may specify the Anchor portion in any of the following formats

- \(<YYYY><MM><DD>\) format. The valid range is from 19000101 to 99991231.
- The symbol \(ED\) is only valid in a start date and represents the supplied end date anchor.
- The symbol \(SD\) is only valid in an end date and represents the supplied start date anchor.
- \(<C><U><n><YYYY>, where:
  - \(<C>\) represents the calendar type, which can be either \(C\) (calendar) or \(F\) (fiscal).
  - \(<U>\) represents the period unit, which can be either \(Q\) (quarterly), \(S\) (semi-annually) or \(Y\) (yearly).
  - \(<n>\) represents a valid integer value for the specified period unit. So, for Quarterly, \(<n>\) must be either 1, 2, 3, or 4. For Semi-annually, \(<n>\) must be either 1 or 2. For Yearly, \(<n>\) must be 1 or it may be omitted.
  - \(<YYYY>\) represents the year. The valid range is from 1900 to 9999.

**Offset**

If you supply an offset it must always be in the form \(+[-]>[n]<C><U>\), where:

- The first character is always a plus (+) or minus (−) sign to indicate the direction of the offset from the Anchor date.
- The second character (\(<n>\)) is an optional multiplier. It must be between 0 and 32767 and the default if it is not specified is 0.
The third character, <C> is either A (actual), C (calendar) or F (fiscal).

- For Actual or Calendar types the fourth character, <U> is either D (daily), W (weekly), M (monthly), Q (quarterly), S (semi-annually), or Y (yearly).

- For Fiscal calendar types the fourth character, <U>, is either Q (quarterly), S (semi-annually) or Y (yearly).

If you use the Actual calendar type, the offset is applied precisely with no "rounding". For example, +2AW from a Tuesday will result in the Tuesday two weeks hence. +1AM from the 16th will result in the 16th of the following month.

If you use the Calendar or Fiscal calendar types, the resulting date is rounded down to the last active date of the previous period. For example, +1CW from a Tuesday will result in the Friday of the same week, +1CM from the 16th will result in the last active day of that month, +CM from the 16th will result in the last active day of the previous month.

If the multiplier is not specified and defaults to 0 the resulting date will be the same as the Anchor if the Actual calendar type is used. If the Anchor is Calendar or Fiscal calendar type then the resulting date will be the end of the prior period.

Examples

- 20080409 represents 9 April 2008.
- CQ42007 represents 31 December 2007.
- 20080409-1AM represents 9 March 2008 - exactly one month previous to the anchor.
- 20080409-1CM represents 29 February 2008 - the end of the month prior to 9 March 2008.
- A start date of 20080409-3CM and an end date of 20080409-CM will provide a range that covers the three calendar months prior to the anchor date of 9 April 2008 (that is, January, February and March).
- -3CQ evaluated on 23 June 2008 represents 29 June 2007 (because 30 June 2007 was a Saturday).
- A start date of 20080409-2AQ and an end date of SD+1AD represents a range from 9 October 2007 to 10 April 2008 (Note that the SD refers only to the Anchor part of the start date not the result after adding the offset to the Anchor).

7.2 Reference Data Service

The reference data service provides the ability to access the following Bloomberg data with the request/response paradigm:

- Reference Data Request
  A Reference Data Request provides a snapshot of the current value of a security/field pair.

- Historical End-of-Day Data
  A Historical Data Request provides end-of-day data over a defined period of time for a security/field pair.
Historical Intraday Ticks
An Intraday Tick Request provides each tick over a defined period of time for a security and event type pair.

Historical Intraday Bars
An Intraday Bar Request provides a series of intraday summaries over a defined period of time for a security and event type pair.

Portfolio Data Request
The Portfolio Data Request enables retrieval of change information and portfolio positions with respect to a specific date in order to see how current market movements have affected user's portfolio's constituent weights.

BEQS (Bloomberg Equity Screening) Request
BEQS (Bloomberg Equity Screening) request returns security data for a selected screen created using the Bloomberg EQS<GO> function.

7.2.1 Reference Data Request and Response Overview

The ReferenceDataRequest enables a snapshot of the current data available for a security/field pair. A list of fields is available via the BLOOMBERG PROFESSIONAL service function FLDS<GO> or using the API fields service. A ReferenceDataRequest must specify at least one or more securities and one or more fields. The API will return data for each security/field pair, or alternatively a message indicating otherwise. This example shows how to construct a ReferenceDataRequest:

```java
Assume we have already opened the //blp/refdata service
Service refDataService = session.getService("//blp/refdata");
Request request = refDataService.createRequest("ReferenceDataRequest");
request.append("securities", "IBM US Equity");
request.append("securities", "/cusip/912828GM6@BGN");
request.append("fields", "PX_LAST");
request.append("fields", "DS002");
d_cid = session.sendRequest(request, null);
```

Response Overview

A PARTIAL_RESPONSE or RESPONSE message will be returned. For large requests, a PARTIAL_RESPONSE will be provided returning part of the information. A RESPONSE
message indicates the request has been fully served. Further information is available in “Appendix A Schemas”. This example shows how to process a ReferenceDataResponse:

```java
private void processReferenceDataResponse(Message msg) throws Exception {
    Element securityDataArray = msg.getElement("securityData");

    for (int i = 0; i < securityDataArray.numValues(); ++i) {
        Element securityData = securityDataArray.getValueAsElement(i);
        System.out.println(securityData.getElementAsString("security"));
        Element fieldData = securityData.get_element("fieldData");

        for (int j = 0; j < fieldData.numElements(); ++j) {
            Element field = fieldData.getElementAt(j);
            System.out.println(field.name() + " = " + field.getValueAsString());
        }

        System.out.println("\n");
    }
}
```

### 7.2.2 Historical Data Request

The HistoricalDataRequest enables the retrieval of end-of-day data for a set of securities and fields over a specified period, which can be set to daily, monthly, quarterly, bi-annually or annually. At least one security and one field are required, along with start and end dates. There are a range of options that can be specified in the request, which are outlined in “Appendix A Schemas”. This example shows how to construct a HistoricalDataRequest for monthly last price data for 2010:

```java
Service refDataService = session.getService("//blp/refdata");
Request request = refDataService.createRequest("HistoricalDataRequest");
request.append("securities", "IBM US Equity");
request.append("securities", "MSFT US Equity");
request.append("fields", "PX_LAST");
request.append("fields", "OPEN");
request.set("startDate", "20100101");
request.set("endDate", "20101231");
request.set("periodicitySelection", "MONTHLY");
```
Response Overview

A successful HistoricalDataResponse holds information on a single security. It contains a HistoricalDataTable with one HistoricalDataRow for each interval returned.

```java
private void processHistoricalDataResponse(Message msg) throws Exception {
    Element securityData = msg.getElement("securityData");
    Element fieldDataArray = securityData.getElement("fieldData");
    for (int j = 0; j < fieldDataArray.numValues(); ++j) {
        Element fieldData = fieldDataArray.getValueAsElement(j);
        for (int k = 0; k < fieldData.numElements(); ++k) {
            Element field = fieldData.getElementAt(k);
            System.out.println("\t" + field.name() + " = "
                                + field.getValueAsString());
        }
    }
}
```

7.2.3 Intraday Tick Request

Bloomberg maintains a tick-by-tick history going back 140 days for all securities where streaming data is available. This intraday data can be used to draw detailed charts, for technical analysis, or to retrieve the initial data for a monitoring graph function such as the GIP<GO> function on the BLOOMBERG PROFESSIONAL service.

The IntradayTickRequest enables retrieval of tick-by-tick history for a single security. In addition, the event type(s) and date/time start- and end-points in UTC must be specified.

This example shows how to construct an IntradayTickRequest:

```java
Service refDataService = session.getService("//blp/refdata");
Request request = refDataService.createRequest("IntradayTickRequest");
request.set("security", "VOD LN Equity");
request.append("eventTypes", "TRADE");
request.append("eventTypes", "AT_TRADE");
request.set("startDateTime", new Datetime(2010, 07, 26, 10, 30, 0, 0));
request.set("endDateTime", new Datetime(2010, 07, 26, 14, 30, 0, 0));
```

Response Overview

A successful IntradayTickResponse will contain an array of IntradayTickData providing information on each tick in the specified time range. The time taken to respond to this request
7.2.4 Intraday Bar Services

Bloomberg maintains a tick-by-tick history going back 140 days for all securities where streaming data is available. This intraday data can be used to draw detailed charts, for technical analysis, or to retrieve the initial data for a monitoring graph function such as the GIP function on the BLOOMBERG PROFESSIONAL service.

The Intraday Bar Request enables retrieval of summary intervals for intraday data covering five event types, TRADE, BID, ASK, BEST_BID, and BEST_ASK, over a period of time. Note that only one event type can be specified per request.

Each bar contains OPEN, HIGH, LOW, CLOSE, VOLUME, and NUMBER_OF_TICKS. The interval size of the bars can be set to as low as 1 minute and to as high as 1440 minutes (24 hours).

Each IntradayBarRequest can only submit one single instrument. In addition, the event type, interval, and date/time start and end-points in UTC must be specified. This example shows how to construct an IntradayBarRequest.

```java
private void processIntradayTickResponse(Message msg) throws Exception {
    Element data     = msg.getElementById("tickData").getElement("tickData");
    int     numItems = data.numValues();

    for (int i = 0; i < numItems; ++i) {
        Element item   = data.getValueAsElement(i);
        Datetime time  = item.getElementById("time");
        String   type  = item.getElementById("type");
        double   value = item.getElementById("value");
        int      size  = item.getElementById("size");
        String   cc;
        if (item.hasElement("conditionCodes")) {
            cc = item.getElementById("conditionCodes");
        }
        Process values
    }
}
```

```java
Service refDataService = session.getService("//blp/refdata");
Request request = refDataService.createRequest("IntradayBarRequest");
request.set("security", "IBM US Equity");
request.set("eventType", "TRADE");
request.set("interval", 60); // bar interval in minutes
request.set("startDateTime", new Datetime(2010, 03, 26, 13, 30, 0, 0));
request.set("endDateTime", new Datetime(2010, 03, 26, 21, 30, 0, 0));
```
A successful **IntradayBarResponse** will contain an array of **BarTickData** each of which contains open, high, low, close, number of events and volume values. Further information is available in "Appendix A Schemas". This example shows how to interpret an **IntradayBarResponse**.

```java
private void processIntradayBarResponse(Message msg) throws Exception {
    Element data = msg.getElement("barData").getElement("barTickData");
    int numBars = data.numValues();
    for (int i = 0; i < numBars; ++i) {
        Element bar = data.getValueAsElement(i);
        Datetime time = bar.getElementAsDate("time");
        double open = bar.getElementAsFloat64("open");
        double high = bar.getElementAsFloat64("high");
        double low = bar.getElementAsFloat64("low");
        double close = bar.getElementAsFloat64("close");
        int numEvents = bar.getElementAsInt32("numEvents");
        long volume = bar.getElementAsInt64("volume");
        Process values
    }
}
```

### 7.2.5 Portfolio Data Request

The PortfolioDataRequest enables retrieval of change information and portfolio positions with respect to a specific date in order to see how current market movements have affected their portfolio's constituent weights.

**Note:** The user's portfolio is identified by its Portfolio ID, which can be found on the upper right hand corner of the toolbar on the portfolio's **PRTUGO** page. This information can also be accessed historically by using the **REFERENCE_DATE** override field and supplying the date in 'YYYYMMDD' format.

**Response Overview**

A **PARTIAL_RESPONSE** or **RESPONSE** message will be returned. For large requests a **PARTIAL_RESPONSE** will be provided returning part of the information. A **RESPONSE** message indicates the request has been fully served. Further information is available in "Appendix A Schemas".

### 7.2.6 BEQS Request

BEQS (Bloomberg Equity Screening) request returns security data for a selected screen created using the Bloomberg EQS Terminal function.
7.3 Market Data Service

The Market Data service enables retrieval of streaming data for securities which are priced intraday, by using the API subscription paradigm. Update messages are pushed to the subscriber once the field value changes at the source. These updates can be real time or delayed, based upon the requestor’s exchange entitlements or through setting a delayed subscription option. All fields desired must explicitly be listed in the subscription to receive their updates.

Response Overview

Once a subscription is established, the stream will supply messages in SUBSCRIPTION_DATA events. The initial message returned, known as a "SUMMARY" message, will contain a value for all the fields specified in the subscription. Subsequent messages may contain values for some or all of the requested Bloomberg fields. It is possible that a message contains none of the requested Bloomberg fields as the messages are only filtered based on the fields they could contain rather than the fields they actually contain and many fields in the streaming events are optional. The Bloomberg API will ensure all messages that contain any of the fields you have explicitly subscribed for are pushed to your application. Finally the stream may return additional fields in these messages, for which were not included in the subscription. These additional fields are not filtered for the purpose of speed, and their inclusion is subject to change at any time.

Some of the fields that are returned also have a null state. For example the fields BID and ASK have values of type float and usually give positive values that you can use to populate your own caches. However there are times when these fields will be set to a null value. In the case of BID and ASK fields this is usually interpreted as an instruction to clear the values in your caches. Therefore it is important to test to see if the field is null before you try and retrieve a value from it.

This example shows how to subscribe for streaming data.

```java
Assume that session already exists and the "/blp/mktdata" service has been successfully opened.
SubscriptionList subscriptions = new SubscriptionList();
subscriptions.add("IBM US Equity",
   "LAST_PRICE,BID,ASK",
   "");
subscriptions.add("/cusip/912828GM6@BGN",
   "LAST_PRICE,BID,ASK,BID_YIELD,ASK_YIELD",
   ");
session.subscribe (subscriptions);
```
7.4 Custom VWAP Service

The Custom Volume Weighted Average Price (VWAP) Service provides streaming VWAP values for equities. This service allows for a customized data stream with a series of overrides which are documented in “Appendix A.5 Schema for Market Data and Custom VWAP”.

```
Assume that session already exists and the "/blp/mktvwap" service has been successfully opened.
SubscriptionList subscriptions = new SubscriptionList();
subscriptions.add("/blp/mktvwap/ticker/IBM US Equity" +
  "?VWAP_START_TIME=10:00&VWAP_END_TIME=16:00",
  "LAST_PRICE,BID,ASK",
  ");
session.subscribe(subscriptions);
```

Response Behavior

The response will return a message containing a selection of VWAP fields.

7.5 Market Bar Subscription Service

The Market Bar Service is subscription based service that provides streaming (real time and delayed) intraday bars. This service allows for bucketized data stream where each bucket ("bar") will consist of the following aspect fields:

- time
- low
- value
- open
- close
- volume
- high
- number of ticks
- datetime

The major advantage of the service is for clients wishing to retrieve HIGH/LOW prices for a specified time interval in streaming format. A subscription to a market bar requires the service to be explicitly specified in the topic.

Topic String:

"/BLP/MKTBAR/SYMOLOGY/SECURITY?START_TIME=st&END_TIME=et&BAR_SIZE=bz"

For example:

"/blp/mktbar/ticker/VOD LN Equity?start_time=9:30&bar_size=10"
MKTBAR service is based on TRADE ticks only. Hence, the subscription topic string should have the option “fields=LAST_PRICE”. The following code snippet shows a subscription to market bars:

```java
Assume that the blp/mktbar service has already been opened successfully. SubscriptionList d_subscriptions = new SubscriptionList();
d_subscriptions.add(new Subscription(
        "//blp/mktbar/TICKERX/IBM US Equity",
        "last_price",
        "bar_size=5&start_time=13:30&end_time=20:00",
        new CorrelationID("IBM US Equity")));
d_session.subscribe(d_subscriptions);
```

### Response Behavior

Successful subscription to MKTBAR service will result in the following types of messages being sent to subscriber:

- **MarketBarStart**
- **MarketBarUpdate**
- **MarketBarIntervalEnd**
- **MarketBarEnd**

**MarketBarStart** is generated upon every new bar; therefore the frequency of this event will depend upon the bar_size setting and the fact that security is active at the time. A **MarketBarStart** event will return all fields of the bar with values filled in since the start if the bar until subscription time. (See “A.4 Market Bar Subscription” on page 198.) Subsequently, on every TRADE update a MarketBarUpdate will be sent.

**MarketBarUpdate** will only include fields that have updated since the bar start or last update. Fields that always update are VALUE, VOLUME, NUMBER_OF_TICKS, and CLOSE.

**MarketBarIntervalEnd** is sent at the end of each bar and will always precede next MarketBarStart. This message only contains TIME and DATE.

**NOTE:** MarketBarIntervalEnd is sent consistently at the end of each bar interval even if there are no TRADEs for the security at the moment.

**MarketBarEnd** only occurs when the last market bar has been received, i.e., the end_time has been reached. This message only contains TIME and DATE.

Please note there is no initial summary returned for streaming intraday bars for start date earlier than now. Reference data intraday bar request before a subscription will be required to get an initial snapshot if needed.

When a market bar subscription is set to return delayed data, the market bar start message will not be returned until the delayed period has passed.
7.6 API Field Information Service

The Field Information service provides details and a search capability on fields in the Bloomberg data model using the API request/response paradigm. Information can be retrieved in three ways:

- **Field Information Request**
  A Field Information Request provides a description on the specified fields in the request.

- **Field Search Request**
  A Field Information Request provides the ability to search the Bloomberg data model with a search string for field mnemonics.

- **Categorized Field Search Request**
  A Categorized Field Search Request provides the ability to search the Bloomberg data model based on categories with a search string for field mnemonics.

7.6.1 Field Information Request

A `FieldInfoRequest` returns a description for the specified fields included in the request. The request requires one or more fields specified as either a mnemonic or an alpha-numeric identifier. It is also possible to specify in the request to return the documentation as per `FLDS<GO>`. This example shows how to construct a `FieldInfoRequest`.

```java
Service fieldInfoService = session.getService("//blp/apiflds");
Request request = fieldInfoService.createRequest("FieldInfoRequest");
request.append("id", "LAST_PRICE");
request.append("id", "pq005");
request.append("id", "ds002");
request.set("returnFieldDocumentation", true);
request.append("properties", "fieldoverridable");
```
Response Behavior

A successful FieldResponse will contain an array of FieldData. The FieldData contains the field's unique id and information about the field. This example shows how to process a single FieldResponse.

```java
private void processFieldResponse(Message msg) throws Exception {
    Element fieldDataArray = msg.getElement("fieldData");
    for (int i = 0; i < fieldDataArray.numValues(); ++i) {
        Element fieldData = fieldDataArray.getValueAsElement(i);
        Element fieldInfo = fieldData.getElementById("fieldInfo");
        System.out.println(
            fieldData.getElementByIdAsString("id") + " " +
            fieldInfo.getElementByIdAsString("mnemonic") + " (" +
            fieldInfo.getElementByIdAsString("description") + ") " +
            fieldInfo.getElementByIdAsString("datatype");
        }
    }
}
```

7.6.2 Field Search Request

A FieldSearchRequest returns a list of fields matching a specified search criterion. The request specifies a search string and it may also contain criteria used to filter the results. This criterion allows for the filtering by category, product type and field type. Detailed information on these settings is located in "Appendix A Schemas". This example shows how to construct a FieldSearchRequest.

```java
Service fieldInfoService = session.getService("//blp/apiflds");
Request request = fieldInfoService.createRequest("FieldSearchRequest");
request.set("searchSpec", "last price");
Element exclude = request.getElementById("exclude");
exclude.setElement("fieldType", "Static")
```

Response Behavior

A FieldSearchRequest returns a FieldResponse just as a FieldInfoRequest does.

7.6.3 Categorized Field Search Request

A CategorizedFieldSearchRequest returns a list of fields matching a specified search criterion. The request specifies a search string and it may also contain criteria used to filter the results. This criterion allows for the filtering by category, product type and field type.
Detailed information on these settings is located in "Appendix A Schemas". This example shows how to construct a CategorizedFieldSearchRequest.

```java
Service fieldInfoService = session.getService("//blp/apiflds");
Request request = fieldInfoService.createRequest(
    "CategorizedFieldSearchRequest");
request.set("searchSpec", "last price");
```

**Response Behavior**

A successful CategorizedFieldResponse will contain an array of CategoryData that contains a flattened representation of the matching fields arranged by the category tree. This example shows how to process a single CategorizedFieldResponse.

```java
private void processCategorizedFieldResponse(Message msg) throws Exception {
    Element categoryArray = msg.getElement("category");
    for (int i = 0; i < categoryArray.numValues(); ++i) {
        Element categoryData = categoryArray.getValueAsElement(i);
        System.out.println("Category:" + categoryData.getElementAsString("categoryName"));
        Element fieldDataArray = categoryData.getElement("fieldData");
        for (int j = 0; j < fieldDataArray.numValues(); ++j) {
            Element fieldData = fieldDataArray.getValueAsElement(i);
            Element fieldInfo = fieldData.getE
```
7.7 Page Data Service

The Page Data service of the API provides access to GPGX pages and the data they contain. This is a subscription service, where the GPGX number, the monitor number, the page number and the required rows (fields) must be provided.

The topic is constructed as follows:

0708/012/0001

where:

0708 is the GPGX number
012 is the monitor number
0001 is the page number

An array of strings is used to specify the rows on the page that are of interest. These can be specified as individual rows, multiple rows separated by commas, or ranges of rows, as follows:

<table>
<thead>
<tr>
<th>String</th>
<th>Rows Specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1&quot;</td>
<td>The first row on the page</td>
</tr>
<tr>
<td>&quot;1,2,3&quot;</td>
<td>Rows 1,2 and 3 on the page</td>
</tr>
<tr>
<td>&quot;1,6-10,15,16&quot;</td>
<td>Row 1, rows 6 to 10 and rows 15 and 16</td>
</tr>
</tbody>
</table>

The following example shows how to create a subscription, and demonstrates how the subscription fields are used to pass the rows the user wants to subscribe to.

```csharp
String topic = "0708/012/0001"
List<string> fields = new List<string>();
fields.Add("15-18");  // subscribing to rows 15 to 18
subscriptions.Add(new Subscription("//blp/pagedata/" + topic,
                                   fields,
                                   null,
                                   new CorrelationID(topic)));
```

Response Behaviour

Once a subscription has been created, and the subscription status messages have been processed, two event types might be received:

PageUpdate

A PageUpdate event contains a current view of the entire page. It provides the dimensions of the page, followed by a rowUpdate element for each row on the page. A full page update will
be received first (all the rows on the page), regardless of the requested rows, and acts as an initial paint of the page, prior to receiving ongoing updates.

```json
PageUpdate = {
    numRows = 23
    numCols = 80
    rowUpdate[] = {
        rowUpdate = {
            rowNum = 1
            spanUpdate[] = {
                spanUpdate = {
                    startCol = 1
                    length = 80
                    text =
                        attr[] = {
                        }
                    fgColor = DARKBLUE
                    bgColor = WHITE
                }
            }
        }
        .
        .
        rowUpdate = {
            rowNum = 23
            spanUpdate[] = {
                spanUpdate = {
                    startCol = 1
                    length = 80
                    text =
                        attr[] = {
                        }
                    fgColor = WHITE
                    bgColor = DARKBLUE
                }
            }
        }
    }
}
```

**RowUpdate**

A RowUpdate event consists of a row number, and one or more spanUpdate elements. Each spanUpdate element describes the location and size of the data (startCol, length), the data itself (text), any attributes associated with that piece of data, and the foreground and background colors. The RowUpdate event is structured in exactly the same way as the rowUpdate element of the PageUpdate event.
RowUpdate = {
    rowNum = 15
    spanUpdate[] = {
        spanUpdate = {
            startCol = 61
            length = 1
            text = 9
            attr[] = {
                fgColor = WHITE
                bgColor = DARKBLUE
            }
        }
    }
}

Possible Attribute Values:

- BLINK
- DOUBLEWIDTH
- INTENSIFY
- POINTANDCLICK
- REVERSE
- UNDERLINE

Possible Color Values for foreground and background:

- AMBER
- BLACK
- DARKBLUE
- DARKGREEN
- DEEPBLUE
- FLASHINGBLUE
- FLASHINGRED
- GRAY
- LIGHTBLUE
- LIGHTGREEN
- ORANGE
- PINK
- RED
- VIOLET
- WHITE
- YELLOW
7.8 Technical Analysis Service

Technical Analysis is a method of evaluating securities by analyzing statistics generated by market activity, such as past prices and volume. Technical analysts do not attempt to measure a security's intrinsic value, but instead use charts and other tools to identify patterns that can suggest future activity. The Technical Analysis Service enables you to download this data and bring it into your application using Bloomberg API.

Table 7-1 details the different Technical Analysis data types:

<table>
<thead>
<tr>
<th>Data Type Description Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Historical End of Day</strong></td>
</tr>
<tr>
<td><strong>Intraday</strong></td>
</tr>
<tr>
<td><strong>Real-time</strong></td>
</tr>
</tbody>
</table>

7.8.1 Historical End of Day study request

The Historical study request enables the retrieval of end-of-day technical analysis data for a specified security and study attributes over the specified time periods of daily, weekly,
monthly, bi-annually and annually. Each Historical study request can submit only a single instrument.

```java
Service tasvcService = session.GetService("//blp/tasvc");
Request request = tasvcService.CreateRequest("studyRequest");
// set security name
request.GetElement("priceSource").GetElement("securityName").SetValue("IBM US Equity");
// set historical price data
request.GetElement("priceSource").GetElement("dataRange").SetChoice("historical");
Element historicalEle = request.GetElement("priceSource").GetElement("historical");
historicalEle.GetElement("startDate").SetValue("20100501"); // set study start date
historicalEle.GetElement("endDate").SetValue("20100528"); // set study end date
// DMI study example - set study attributes
request.GetElement("studyAttributes").SetChoice("dmiStudyAttributes");
Element dmiStudyEle = request.GetElement("studyAttributes");
dmiStudyEle.GetElement("period").SetValue(15); // DMI study interval
// set historical data price sources for study
dmiStudyEle.GetElement("priceSourceLow").SetValue("PX_LOW");
dmiStudyEle.GetElement("priceSourceClose").SetValue("PX_LAST");
```

**Response Behaviour**

A successful studyResponse holds information on the requested security. It contains a studyDataTable with one studyDataRow for each interval returned.
7.8.2 Intraday bar study request

The Intraday Bar type study request enables the retrieval of summary intervals of intraday technical analysis data for a specified study attributes for five event types, TRADE, BID, ASK, BEST_BID, and BEST_ASK, over a period of time. Each Intraday study request can only submit only a single instrument. In addition, the event type, interval and date/time start and end-points in UTC must be specified.
Response Behaviour

A successful studyResponse holds information on the requested security. It contains a studyDataTable with one studyDataRow for each bar interval returned.

Service tasvcService = session.GetService("//blp/tasvc");
Request request = tasvcService.CreateRequest("studyRequest");
// set security name
request.GetElement("priceSource").GetElement("securityName").SetValue("IBM US Equity");
Element intradayEle = request.GetElement("priceSource").GetElement("dataRange").GetElement("intraday");
// set intraday price data
intradayEle.GetElement("eventType").SetValue("TRADE"); // intraday event type
intradayEle.GetElement("interval").SetValue(60); // intraday interval
intradayEle.GetElement("startDate").SetValue("2010-05-26T13:30:00"); // set study start date
intradayEle.GetElement("endDate").SetValue("2010-05-27T13:30:00"); // set study end date
// smavg study example - set study attributes
request.GetElement("studyAttributes").SetChoice("smavgStudyAttributes");
Element smavgStudyEle = request.GetElement("studyAttributes").GetElement("smavgStudyAttributes");
smavgStudyEle.GetElement("period").SetValue(15); // SMAVG study interval
smavgStudyEle.GetElement("priceSourceClose").SetValue("close");
7.8.3 Real time study request

The real time study request provides the ability to subscribe to real time technical analysis data points for a specified study field attributes and period. Each real time study subscription can only subscribe to a single study field.

Assume that session already exists and the "/blp/tasvc" service has been successfully opened.

```csharp
SubscriptionList subscriptions = new SubscriptionList();
subscriptions.Add(new Subscription("/blp/tasvc/ticker/IBM US Equity?fields=WLPR&" +
  "priceSourceClose=LAST_PRICE&" +
  "priceSourceHigh=HIGH&" +
  "priceSourceLow=LOW&" +
  "periodicitySelection=DAILY&" +
  "period=14", new CorrelationID("IBM US Equity_WLPR"));
session.subscribe (subscriptions);
```
Response Behaviour

Once a subscription is established, the stream will supply messages in SUBSCRIPTION_DATA events. In addition to the study field subscribed, you may receive additional study fields in these messages which were not subscribed. These additional fields are not filtered for the purpose of speed and their inclusion is subject to change at any time.

7.9 API Authorization

The Authorization service enables an application to handle the Bloomberg concept of Permissioning, by checking authorization and entitlement through the creation of Identities which represent users and/or applications. These Identities contain the entitlement identifiers for data enabled under the user/application. The entitlements are then used in combination with those retrieved from market or reference data to decide whether the entity is allowed to view the data. Detailed documentation is provided in "Authorization and Permissioning Systems" on page 54.

Response Behaviour

The response message indicates a pass or fail.

7.10 Instruments Service

The Instruments Service (//blp/instruments) is used to perform three types of operations. The first is a Security Lookup Request, the second is a Curve Lookup Request and the third is a Government Lookup Request. These three operations are covered in the following sections.

<table>
<thead>
<tr>
<th>Request</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Lookup Request</td>
<td>InstrumentListRequest Operation</td>
</tr>
<tr>
<td>Curve Lookup Request</td>
<td>CurveListRequest Operation</td>
</tr>
<tr>
<td>Government Lookup Request</td>
<td>GovtListRequest Operation</td>
</tr>
</tbody>
</table>

7.10.1 Security Lookup Request

The Security Lookup (a.k.a. Instrument Lookup) request constructs a search based upon the "query" element's string value, as well as the additional filters that you set, such as the yellow key and language override elements. This functionality can also be found on the Bloomberg Professional service using the SECF <GO> function. By setting the language override element, you will obtain your results translated into that specified language.
The following code snippet demonstrates how to make a security lookup request and assumes that a session already exists and that the "/blp/instruments" service has been successfully opened.

```java
Service secfService = session.getService("/blp/instruments");
Request request = secfService.createRequest("instrumentListRequest");
request.asElement().setElement("query", "IBM");
request.asElement().setElement("yellowKeyFilter", "YK_FILTER_CORP");
request.asElement().setElement("languageOverride", "LANG_OVERRIDE_NONE");
request.asElement().setElement("maxResults", 10);
sendRequest(request, session);
```

**Figure 7-1: C++ code snippet - constructing a security lookup request**

7.10.2 Curve Lookup Request

The Curve Lookup request can retrieve a curve based on its country code, currency code, type, subtype, curve specific ID, and the Bloomberg ID for that curve.

The following code snippet demonstrates how to make a curve lookup request and assumes that a session already exists and that the "/blp/instruments" service has been successfully opened.

```java
Service curveService = session.getService("/blp/instruments");
Request request = curveService.createRequest("curveListRequest");
request.asElement().setElement("query", "GOLD");
request.asElement().setElement("bbgid", "YCCD1016");
request.asElement().setElement("countryCode", "US");
request.asElement().setElement("currencyCode", "USD");
request.asElement().setElement("curveid", "CD1016");
request.asElement().setElement("type", "CORP");
request.asElement().setElement("subtype", "CDS");
request.asElement().setElement("maxResults", "10");
sendRequest(request, session);
```

**Figure 7-2: C++ code snippet - constructing a curve lookup request**

7.10.3 Government Lookup Request

The Government lookup does a search through government securities. As with every type of request, you can specify the 'query' string and the maximum number of results. And, since every government security has a ticker that is not unique, you can also filter these securities by this ticker. For example, you can specify filter tickers that are equal to "T" or set Partial Match (i.e., "partialMatch") to true and filter out all government securities beginning with the letter "T". You would do this by setting the "query" element value to "T**."
The following code snippet demonstrates how to make a government lookup request and assumes that a session already exists and that the "/blp/instruments" service has been successfully opened.

```c++
Service govtService = session.getService("/blp/instruments");
Request request = govtService.createRequest("govtListRequest");
request.asElement().setElement("partialMatch", true);
request.asElement().setElement("query", "T*");
request.asElement().setElement("ticker", "LANG_OVERRIDE_NONE");
request.asElement().setElement("maxResults", 10);
sendRequest(request, session);
```

**Figure 7-3: C++ code snippet - constructing a government lookup request**

### 7.10.4 Response Behaviors

Each lookup response will comprise of zero, or more, PARTIAL_RESPONSE event types and one RESPONSE event type event, which you will be familiar with if you have developed Bloomberg API applications using any of the other request/response services, such as //blp/refdata, //blp/apiflds or //blp/tasvc.

The following C++ code demonstrates how to handle the response for each of the three types of requests:

```c++
void dumpInstrumentResults(const std::string& msgPrefix, const Message& msg) {
    const Element& response = msg.asElement();
    const Element& results = response.getElement("results");
    std::cout << ">>> Received " << results.numValues() << " elements" << std::endl;

    size_t numElements = results.numValues();
    std::cout << msgPrefix << " " << numElements << " results:
    for (size_t i = 0; i < numElements; ++i) {
        Element result = results.getValueAsElement(i);
        std::cout << std::setw(2) << (i + 1) << " : " << std::setw(30)
                    << result.getElementAsString("security")
                    << " - "
                    << result.getElementAsString("description")
                    << std::endl;
    }
}
```

**Figure 7-4: Handling a Security Lookup Request**
void dumpCurveResults(const std::string& msgPrefix, const Message& msg) {
    const Element& response = msg.asElement();
    const Element& results = response.getElement("results");
    std::cout << ">>> Received " << results.numValues() << " elements" << std::endl;

    size_t numElements = results.numValues();

    std::cout << msgPrefix << ' ' << numElements << " results:" << std::endl;
    for (size_t i = 0; i < numElements; ++i) {
        Element result = results.getValueAsElement(i);
        std::cout << std::setw(2) << (i + 1) << " : " << std::setw(30)
               << " - '" << result.getElementAsString("description") << "' "
               << "country=" << result.getElementAsString("country") << " "
               << "currency=" << result.getElementAsString("currency") << " "
               << "curveid=" << result.getElementAsString("curveid") << " "
               << "type=" << result.getElementAsString("type") << " "
               << "subtype=" << result.getElementAsString("subtype") << " "
               << "publisher=" << result.getElementAsString("publisher") << " "
               << "bbgid=" << result.getElementAsString("bbgid")
               << std::endl;
    }
}

Figure 7-5: Handling a Curve Lookup Request
7.10.5 Code Example

We have created one example, listed below, to demonstrate all three of the lookup operations, which can be found in the C++, Java, and .NET SDK example folders.

**SecurityLookupExample** - This example demonstrates how to make a security, curve and government lookup request using the //blp/instruments service.

```cpp
void dumpGovtResults(const std::string& msgPrefix, const Message& msg) {
    const Element& response = msg.asElement();
    const Element& results = response.getElement("results");
    std::cout << ">>> Received " << results.numValues() << " elements" << std::endl;

    size_t numElements = results.numValues();

    std::cout << msgPrefix << ' ' << numElements << " results:" << std::endl;
    for (size_t i = 0; i < numElements; ++i) {
        Element result = results.getValueAsElement(i);
        std::cout << std::setw(2) << (i + 1) << " " << std::setw(30)
                  << result.getElementAsString("parseky")
                  << ", "
                  << result.getElementAsString("name")
                  << " - "
                  << result.getElementAsString("ticker")
                  << std::endl;
    }
}
```

Figure 7-6: Handling a Government Lookup Request
8 Publishing

8.1 Overview

The Bloomberg API allows customer applications to publish data as well as consume it. Customer data can be published for distribution within the customer’s enterprise, contributed to the Bloomberg infrastructure, distributed to others, or used for warehousing.

Publishing applications might simply broadcast data or they can be “interactive”, responding to feedback from the infrastructure about the currently active subscriptions from data consumers. This chapter will illustrate both paradigms.

8.2 The Programming Examples

The two examples explored in this chapter are BroadcastOneTopic.cpp and InteractivePublisher.cpp.

8.3 Simple Broadcast

In a simple broadcast, the publishing application sends data but has no indication if anyone is consuming that data. In this simple example, data will be produced for a single topic. The major stages are:

- Creating a session.
- Obtaining authorization.
- Creating the topic.
- Publishing events for the topic to the designated service.

Each of these stages will now be examined in detail.

8.3.1 Creating a Session

Sessions for publication are created in the same manner as those for consuming data. The key difference is that they are managed by an instance of ProviderSession instead of Session.
The event handler plays no significant role in this example and will not be examined.

### 8.3.2 Authorization

The authorization stage, if successful, provides a valid *Identity* object which is required for later operations. Authorization is done by the "//blp/apiauth" service on receipt of an authorization request.

See for *"Authorization and Permissioning Systems" on page 54* details.
Name TOKEN("token");
Name TOKEN_SUCCESS("TokenGenerationSuccess");
Name TOKEN_FAILURE("TokenGenerationFailure");
Name AUTHORIZATION_SUCCESS("AuthorizationSuccess");
EventQueue tokenEventQueue;
session.generateToken(CorrelationId(), &tokenEventQueue);
std::string token;
Event event = tokenEventQueue.nextEvent();
if (event.eventType() == Event::TOKEN_STATUS) {
    MessageIterator iter(event);
    while (iter.next()) {
        Message msg = iter.message();
        msg.print(std::cout);
        if (msg.messageType() == TOKEN_SUCCESS) {
            token = msg.getElementAsString(TOKEN);
        } else if (msg.messageType() == TOKEN_FAILURE) {
            break;
        }
    }
}
if (token.length() == 0) {
    std::cout << "Failed to get token" << std::endl;
}

session.openService("/blp/apiauth");
Service authService = session.getService("/blp/apiauth");
Request authRequest = authService.createAuthorizationRequest();
authRequest.set(TOKEN, token.c_str());

EventQueue authQueue;
Identity providerIdentity = session.createIdentity();
session.sendAuthorizationRequest( authRequest, &providerIdentity, CorrelationId(), &authQueue);
8.3.3 Creating a Topic

Before publishing data, the application must create a Topic object on the appropriate service. This example uses synchronous method `createTopics()` of the `ProviderSession` to create a Topic on `//blp/test` service from a topic string "testtopic".

```java
else if (event.eventType() == EventType.RESPONSE
    || event.eventType() == EventType.PARTIAL_RESPONSE
    || event.eventType() == EventType.REQUEST_STATUS) {
    for (Message msg: event) {
        if (msg.correlationID().equals(d_authorizationResponseCorrelationId)) {
            Object authorizationResponseMonitor =
                msg.correlationID().object();
            synchronized (authorizationResponseMonitor) {
                if (msg.messageType() == AUTHORIZATION_SUCCESS) {
                    d_authorizationResponse = Boolean.TRUE;
                    authorizationResponseMonitor.notifyAll();
                } else if (msg.messageType() == AUTHORIZATION_FAILURE) {
                    d_authorizationResponse = Boolean.FALSE;
                    System.err.println("Not authorized: " +
                        msg.getElement("reason");
                } else {
                    assert d_authorizationResponse == Boolean.TRUE;
                    System.out.println("Permissions updated");
                }
            }
        }
    }
}
```
8.3.4 Publishing

In this example, data is published by sending events to the designated service, "/blp/test". Event objects are obtained from the service and populated with the topic and the application specific data. In this simple example, each event contains a single data message; however, in general, each event can contain multiple messages.

In this simple example, the data is just an integer value that is incremented and published every ten seconds.

```cpp
... ...
const std::string myService = "//blp/test";
const std::string myTopic = "testtopic";
TopicList topicList;
topicList.add((myService + "/ticker/" + myTopic).c_str(),
              CorrelationId((long long)1));

session.createTopics(
    &topicList,
    ProviderSession::AUTO_REGISTER_SERVICES,
    providerIdentity);

Topic topic;
for (size_t i = 0; i < topicList.size(); ++i) {
    if (topicList.statusAt(i) == TopicList::CREATED) {
        topic = session.getTopic(topicList.messageAt(i));
    }
}
... ...
```
Interactive Publication

The Bloomberg infrastructure can send events to provider applications when data is needed for a given topic. These events allow the customer applications to "interact" with the Bloomberg infrastructure. Data for a topic need be published only when it is known to have subscribers.

In this simple example, data is published, only as needed, for a set of topics on a single service. The major steps are:

- Creating a session.
- Obtaining authorization.
- Registering for subscription start and stop messages.
- Handling subscription start and stop events, which add and remove topics to the active publication set.
- Creating a topic.
- Publishing events for the active topics of the designated service.

The details for creating a session, obtaining a provider identity, and authorization are the same as in the earlier example; they will not be detailed again.

This design requires the management of a collection of "active" topics for publication. That collection will be populated (and depopulated) by event handling threads and accessed for

```c
...  
Name messageType ("MyMessageType");
Name fieldType ("MyFieldType");

Service service = session.getService(myService.c_str());
for (int value = 1; true; ++value, sleep(10)) {
    Event event = service.createPublishEvent();
    EventFormatter eventFormatter(event);
    eventFormatter.appendMessage(messageType, topic);
    eventFormatter.setElement(fieldName, value);

    session.publish(event);
}

session.stop();
return 0;
}
periodic publication by the main thread. A map will be used to store pairs of topic/CUSIP pairs (keyed on topic). The topics are provided in the start and stop messages, and CUSIPs are obtained by requesting resolution of the received topics.

The multiple threads of this application must not concurrently access the collection; STL containers are not thread-safe in that respect. Since there is only one "reading" thread in this application, a simple mutex suffices. A pthread mutex was chosen because it is familiar to many readers.

```
// InteractivePublisher.cpp
... ...
int main(int argc, char **argv)
{
    Publications activePublications;
    pthread_mutex_t activePublicationsMutex;
    pthread_mutex_init(&activePublicationsMutex, NULL);
    MyEventHandler myEventHandler(&activePublications,
          &activePublicationsMutex);

    SessionOptions sessionOptions;
    sessionOptions.setServerHost("192.168.9.155");
    sessionOptions.setServerPort(8195);
    //sessionOptions.setAuthenticationOptions("AuthenticationType=OS_LOGON");

    sessionOptions.setAuthenticationOptions("AuthenticationMode=APPLICATION_ONLY;
      ApplicationAuthenticationType=APPNAME_AND_KEY;ApplicationName=blp:APP_BBOX");

    ProviderSession session(sessionOptions, &myEventHandler, 0);
    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return -1;
    }
}
```

As we will see later, the event handler is designed to hold pointers to the collection of active topics and to the mutex that manages access to that collection.

### 8.4.1 Registration

On completion of service registration, the application can expect subscription start and subscription stop messages in the context of subscription status events.
8.4.2 Event Handling

The event handler in this example is detailed below. The relevant event type is TOPIC_STATUS. The TOPIC_STATUS event has three message types of interest: TOPIC_CREATED, TOPIC_SUBSCRIBED, and TOPIC_UNSUBSCRIBED.

On receipt of "started" type messages, the event handler adds the topic to a set of topics that require asynchronous topic creation. Once all of the messages in the event have been examined, that list (if non-empty) is sent for resolution. Use of the session’s createTopicsAsync method means that the operation does not block. Rather, the result is returned in a separate event of type TOPIC_CREATED.

When messages indicating successful topic creation are received, the event handler extracts the topic and the corresponding string, creates an item, and adds that item to the collection of active publications. Since a topic may have received a "stop" message while it was being created, there is first a check to see if the topic is still in the "needed" set before it is added to the "active" collection.

On receipt of a "stopped" type, the event handler extracts the topic from the message and deletes the corresponding item in the collection of active publications or the collection of topics needing creation.

Note that all operations use the provided mutex to provide exclusive access for each other.
bool MyEventHandler::processEvent(const Event& event, ProviderSession* session)
{
    switch (event.eventType()) {
    case Event::TOPIC_STATUS:
    {
        TopicList topicList;
        MessageIterator iter(event);
        while (iter.next()) {
            Message msg = iter.message();
            std::cout << msg << std::endl;
            if (msg.messageType() == TOPIC_SUBSCRIBED) {
                Topic topic;
                try {
                    topic = session->getTopic(msg);
                } catch (blpapi::Exception &) {
                }
                if (!topic.isValid()) {
                    topicList.add(msg);
                } else if (d_actPub_p->find(topic) == d_actPub_p->end()) {
                    std::string topicStr = msg.getElementAsString("topic");
                    pthread_mutex_lock(d_actMutex_p);
                    PublicationItem publicationItem(topic, topicStr);
                    d_actPub_p->insert(publicationItem);
                    pthread_mutex_unlock(d_actMutex_p);
                }
            } else if (msg.messageType() == TOPIC_UNSUBSCRIBED) {
                Topic topic;
                try {
                    topic = session->getTopic(msg);
                } catch (blpapi::Exception &) {
                }
                pthread_mutex_lock(d_actMutex_p);
                Publications::iterator it = d_actPub_p->find(topic);
                if (it != d_actPub_p->end()) {
                    d_actPub_p->erase(it);
                }
                pthread_mutex_unlock(d_actMutex_p);
            } catch (blpapi::Exception &) {
            }
        }
    }
}
else if (msg.messageType() == TOPIC_CREATED) {
    try {
        Topic topic = session->getTopic(msg);
        std::string topicStr = msg.getElementAsString("topic");
        pthread_mutex_lock(d_actMutex_p);
        PublicationItem publicationItem(topic, topicStr);
        d_actPub_p->insert(publicationItem);
        pthread_mutex_unlock(d_actMutex_p);
    } catch (blpapi::Exception &e) {
        std::cerr
            << "Exception in Session::getTopic(): "
            << e.description()
            << std::endl;
        continue;
    }
}

if (topicList.size()) {
    session->createTopicsAsync(topicList);
}
} break;
default:
    printMessages(event);
}

return true;

8.4.3 Publication

The publication loop in this example is, in many ways, similar to that used in the first example. There is a value that is incremented every ten seconds and is used to create an event for publication.
Note: The standard C library 'sleep' function is used above. The argument specifies the number of seconds to sleep.

However, there are some differences (highlighted above):

- Rather than a single fixed topic, publication is made for all of the topics in the collection of active publications.
- Note that the mutex is acquired before iterating over that collection.
- There is at most one published event per cycle. Each event may have multiple messages, each with data for a specific topic.
- Although sending an empty event would not be harmful, if the collection of active publications is empty, no event is published for that cycle.
- The published data might vary by topic. Details of the `myValueFor` function are not important and, therefore, not shown.
9 B-Pipe

9.1 Overview

In addition to the core set of services available to licensed users of the Desktop API and Server API products, there is an additional set of services that are offered only to B-Pipe users. The primary purpose of this section is to provide the depth of knowledge required to understanding and utilizing these services in your Bloomberg API application.

They are as follows:

- Market Depth Service (//blp/mktdepth)
- Market List Service (//blp/mktlist)
- Source Reference Service (///blp/srcref)

For information on the core set of services available to B-Pipe users, please see “Core Services” on page 77.

Important Notice

Field filtering is available as a configuration option, which means that B-Pipe clients have the option to change their configurations so that only the fields specified in a subscription are returned. As a result, clients should be able to recognize significant bandwidth savings on their Client LAN.

Contact Bloomberg support to have this feature enabled on your Bloomberg Appliance.

9.2 B-Pipe Services

9.2.1 Market Depth Service

Overview

Market depth, order books and level 2 data are all names for the same set of data. They provide information about the bid and ask prices that currently exist for an instrument.

Generally, the "top of the book", i.e., the price in the top row (row 1) of the order book is also the "best" bid or ask.

Typically the best bid in an order book will be lower than the best ask. This seems natural since people want to buy (bid for) something at a lower price than someone else wants to sell (ask for) the same item. However, it is possible for this situation to become reversed and the best bid price becomes higher than the best ask price. This is known as an inverted or crossed market and can and does occur regularly under specific conditions. The details of the specific conditions vary by market.
Many times exchanges consider order book (level 2) information a separate product from its level 1 data and charge additional fees for access to it. In these cases the level 2 data will have a different EID than the level 1 data. Order books have three characteristics that define them: The number of rows in the book (window size), the type of the order book and the method used to update the book.

There are two types of order books, Market-By-Order (MBO) and Market-By-Level (MBL). An exchange may provide only MBL data, only MBO data or both MBO and MBL data. There are three order book update methods, Replace-By-Level (RBL), Add-Mod-Delete (AMD) and Replace-By-Broker (RBB).

The Market Depth Service

The Market Depth service is subscription-based and allows the subscription to all levels of market depth data. It is available to both BPS (Bloomberg Professional Service) and Non-BPS users.

Before delving into the market depth service and its data, let's first take a look at another way to obtain limited market depth data via the already existing //blp/mktdata service. With this service, you can obtain up to the first 10 levels of market depth by level (aka MBL) data. This is accomplished by making a //blp/mktdata subscription and including one or more of the following fields.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEST_BID1 thru BEST_BID10</td>
<td>First thru tenth best bid price in ten levels of market depth</td>
</tr>
<tr>
<td>BEST_BID1_SZ thru BEST_BID10_SZ</td>
<td>Size of first thru tenth best bid in ten levels of market depth</td>
</tr>
<tr>
<td>BEST_ASK1 thru BEST_ASK10</td>
<td>First thru tenth best ask price in ten levels of market depth</td>
</tr>
<tr>
<td>BEST_ASK1_SZ thru BEST_ASK10_SZ</td>
<td>Size of first thru tenth best ask in ten levels of market depth</td>
</tr>
</tbody>
</table>

For further information regarding making a subscription, please read the “Subscriptions” on page 47.

Keep in mind that this method of obtaining market depth through the //blp/mktdata service is limited to receiving only aggregated Market By Level data for up to 10 levels. This service doesn't allow you to obtain "Market By Order" (MBO) data. Also, the //blp/mktdata service doesn't provide you with information such as the book type or the action performed on that position.

Therefore, if you wish to receive more than 10 levels of market depth by level (MBL) or any market depth by order (MBO) levels, then you will be required to use the //blp/mktdepth
service. Subscribing to this comprehensive service will not only supply you with the order book in its entirety, but also provide you with the book type, action performed, etc.

**Code Examples**

You will find two separate examples in the B-Pipe SDK for C++, Java and .NET. They are as follows:

- **MarketDepthSubscriptionExample**
  This example demonstrates how to make a simple Market Depth subscription for one, or more, securities and display all of the messages to std::cout.

- **MarketDepthSubscriptionSnapshotExample**
  This example demonstrates how to build and update an order and level book. It is comprised of a LevelBook and OrderBook class, which handle the Market Depth By Level and By Order messages, respectively, based upon the returned MD_TABLE_CMD_RT value, and then the main classes which perform the subscription, general message handling and output tasks.

**Number of Rows in an Order Book**

The number of rows in a book may be limited or not. Many exchanges limit their books to as few as 5 rows (positions), others may have as many as 200 rows while still others may not have a predefined limit to the number of rows a book may have. The number of rows that are sent to a client can also be limited by the vendor providing the data. In general, 200 rows are considered a large book. When an order book has a limited size, and most do, prices or orders can be dropped and added back regularly as the top of the book changes. There is no connection between the number of rows in a book and the type and method of the book. Each is independently determined by the source of the book.

**Types of Order Books**

**Market-by-Order (MBO)**

MBO order books show every order that is in the book. If multiple brokers have orders at the same price level the book will show each order, resulting in multiple rows at the same price level. The amount of data that is available at each level varies by the source of the data but it typically consists of the price, size and a broker ID.

**Market-by-Level (MBL)**

MBL order books show only one row for each price. If multiple brokers have bids or asks in at the same price the size of all the brokers orders will be summed and be displayed. Optionally, the number of brokers at that level may also be provided.

The type of an order book is independent of the method used to maintain the order book.
Order Book Methods

Replace-By-Level (RBL)

The first method is called Replace-By-Level (RBL). It is used for both MBO and MBL types of order books. In the RBL method, each row (position) in the order book is directly addressed so that updates to row 1 are specifically addressed to row 1, updates to row 2 are specifically addressed to row 2, etc.

For instance, when a new price is inserted in row 1, the old price that was in row 1 must now be moved to row 2, the price that was in row 2 moved to row 3, etc. This results in multiple messages updating the affected rows in the book. When multiple updates are needed, the MD_MULTI_TICK_UPD_RT field will be present. A non-zero value in this field indicates that additional messages are coming. All related updates must be applied before the book is back in a valid state.

This method works well for small order books, but can become very inefficient for large books, particularly so because a majority of the activity in an order book occurs at the top of the book, requiring frequent retransmission of the entire book. It can also be difficult to know when a single update is complete.

Add-Mod-Del (AMD)

The second order book method is Add-Mod-Delete (AMD). It is used for both MBO and MBL types of order books. The AMD method is much more efficient in sending updates to order books. Instead of addressing each row in the book individually only the changes to the book are sent. This means that client applications must manage any related updates resulting from an Add or Delete event.

For instance, when a new price is inserted at a specific row, the only message sent is the insert. It is the application's responsibility to adjust the position of all the rows that have been shifted down. Likewise, when a row is deleted, it is the application's responsibility to shift all the prices that were below it up. Of course any new price at the bottom of the book requires a separate "Insert", but this is much more efficient than resending the whole book.

The downside of the AMD method is that it depends on receiving and correctly processing every update to keep the book accurate. With the RBL method a missed message will result in the specific row being wrong. But this condition is corrected the next time that row is updated.

Because a single AMD message can affect a single row, one missed message can result in the order book being wrong for the rest of the day or until a recap is sent. Because of this, AMD messages are sent using sequence numbers. If the application detects a gap in the sequence numbers it can recover from the error by re-requesting the entire order book. In other words, resubscribe to the book. If the gap is detected as a result of an issue within the Bloomberg Data Center, Bloomberg will send down an order recap. This form of gap detection is covered in a later section.
**Replace-by-Broker (RBB)**

The third order book method is Replace-By-Broker (RBB). Because it addresses specific broker entries, it is used only for MBO order books. It is a mix of the RBL and AMD methods. It is similar to the RBL method in that each broker's entry is individually addressed. It is similar to the AMD method in that a single update affects the entire book. However, unlike the AMD method, a missed message results in an order book that is wrong only until the next update for that broker.

Both the RBL and AMD methods specify specific row numbers to identify each entry. The RBB method does not use row numbers. Instead the broker code is used to identify the entry.

How RBB order books are sorted is left up to the feed handler. The general rule is to use the price as the primary sort key. The secondary sort key can either be the sequence the orders at the same price were received or an alphabetic listing of all the brokers at the same price.

**Subscribing to Market Depth**

The first step in subscribing to the //blp/mktdepth service is to learn how the subscription strings are formulated. For the string to be valid, you must specify a "type" parameter, which can be either MBO (Market by Order) or MBL (Market by Level). You cannot specify more than one of these in a subscription string. This is appended to the end of the string, immediately following the "?" delimiter.

Here is a list of valid market depth subscription string formats, along with an example of each.

<table>
<thead>
<tr>
<th>Key Field</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomberg Symbol</td>
<td>//blp/mktdepth/bsym/source/symbol</td>
<td>//blp/mktdepth/bsym/LN/VOD?type=MBL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>//blp/mktdepth/bsym/US/AAPL?type=MBO</td>
</tr>
<tr>
<td>Ticker</td>
<td>//blp/mktdepth/ticker/symbol</td>
<td>//blp/mktdepth/ticker/ESM2 Index?type=MBL</td>
</tr>
<tr>
<td>BSID</td>
<td>//blp/mktdepth/bsid/bsid</td>
<td>//blp/mktdepth/bsid/399432473346?type=MBO</td>
</tr>
<tr>
<td>Parsekeyable</td>
<td>//blp/mktdepth/bpkbl/identifier</td>
<td>//blp/mktdepth/bpkbl/QCZ1 Index?type=MBL</td>
</tr>
<tr>
<td>Bloomberg Global ID</td>
<td>//blp/mktdepth/bbgid/source/bbgid</td>
<td>//blp/mktdepth/bbgid/UP/BBG000BH2658?type=MBO</td>
</tr>
</tbody>
</table>

The following code snippet demonstrates how to subscribe for streaming (MBL) market depth data and assumes that a session already exists and that the "//blp/mktdepth" service has been successfully opened.

```c
const char *security = "//blp/mktdepth/isin/US/US4592001014?type=MBL";
SubscriptionList subscriptions;
subscriptions.add(security, CorrelationId((char *)security));
session.subscribe (subscriptions);
```

*Figure 9-1: C++ code snippet: Subscribing for streaming (MBL) market depth data*
Response Overview

The Market Depth response will be a series of SUBSCRIPTION_DATA events, which you will already be familiar with if you have developed Bloomberg API applications using any of the other streaming services, such as //blp/mktdata or //blp/mktvwap.

A SUBSCRIPTION_DATA event message will be of type MarketDepthUpdates, and within each message there will be a MKTDEPTH_EVENT_TYPE and MKTDEPTH_EVENT_SUBTYPE field, along with, possibly, an array of MBO_TABLE_ASK/MBO_TABLE_BID items (for MBO subscription) or MBL_TABLE_ASK/MBL_TABLE_BID (for MBL subscriptions).

The MKTDEPTH_EVENT_TYPE will indicate whether the message is Market by Level (value= MARKET_BY_LEVEL) or Market by Order (value = MARKET_BY_ORDER). Here are the possible values for each MKTDEPTH_EVENT_SUBTYPE:

<table>
<thead>
<tr>
<th>MKTDEPTH_EVENT_SUBTYPE</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_INITPAINT</td>
<td>This is the Initial Paint message for your subscription</td>
</tr>
<tr>
<td></td>
<td>When this message is received, it is an indicator to you to clear the</td>
</tr>
<tr>
<td></td>
<td>book cache and add the rows contained in the message.</td>
</tr>
<tr>
<td></td>
<td>This message will contain the FEED_SOURCE, ID_BB_SEC_NUM_SRC (a.k.a.</td>
</tr>
<tr>
<td></td>
<td>BSID) and MD_BOOK_TYPE. No other messages will contain this</td>
</tr>
<tr>
<td></td>
<td>information, so it is required that you assign a unique correlation</td>
</tr>
<tr>
<td></td>
<td>identifier to each one of your subscriptions in order to map the</td>
</tr>
<tr>
<td></td>
<td>message updates to the initial request.</td>
</tr>
<tr>
<td></td>
<td>For AMD and RBL book types, there will be a WINDOW_SIZE field/value</td>
</tr>
<tr>
<td></td>
<td>pairing, which indicates the number of levels in the book, as</td>
</tr>
<tr>
<td></td>
<td>position is the key to the book. However, this field will not be</td>
</tr>
<tr>
<td></td>
<td>contained in the MBO-RBB initial paint, as the key for this book</td>
</tr>
<tr>
<td></td>
<td>is the broker.</td>
</tr>
<tr>
<td>BID</td>
<td>This indicates a bid quote message</td>
</tr>
<tr>
<td>ASK</td>
<td>This indicates an ask quote message</td>
</tr>
<tr>
<td>BID_RETRANS</td>
<td>In the event of a loss of connectivity upstream, the Bloomberg</td>
</tr>
<tr>
<td></td>
<td>infrastructure will automatically recover (RECAP) and send</td>
</tr>
<tr>
<td></td>
<td>BID_RETRANS and ASK_RETRANS events. Upon receipt of these messages,</td>
</tr>
<tr>
<td></td>
<td>you will receive a CLEARALL message with a MKTDEPTH_EVENT_SUBTYPE</td>
</tr>
<tr>
<td></td>
<td>of RETRANS and you should consider your book in a bad state and</td>
</tr>
<tr>
<td></td>
<td>accept the recovery. Please note that the sequence numbers will be</td>
</tr>
<tr>
<td></td>
<td>set to zero during the recap.</td>
</tr>
<tr>
<td>ASK_RETRANS</td>
<td>See BID_RETRANS description above</td>
</tr>
</tbody>
</table>

Within each TABLE_INITPAINT message you will find one MD_TABLE_CMD_RT field/value pairing for the entire initial paint and then individual MD_TABLE_CMD_RT field/value pairings for each MBL_TABLE_ASK/MBO_TABLE_ASK/ MBL_TABLE_BID/MBO_TABLE_BID that may be present. Thereafter, you will see on MD_TABLE_CMD field/value pairing for each BID or ASK MKTDEPTH_EVENT_SUBTYPE tick update.
The possible string values, which indicate what action should be taken in response to the market depth event, are listed in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNASSIGNED</td>
<td>0</td>
<td>The default constant 'UNASSIGNED' is used to initialize all enumeration type fields</td>
</tr>
<tr>
<td>ADD</td>
<td>1</td>
<td>Add an entry to the order book. When you add this order in the market depth table, you should shift all orders at the market depth position...</td>
</tr>
<tr>
<td>DEL</td>
<td>2</td>
<td>Delete this event from the market depth cache. The delete should occur at the position passed in the market depth event. When cached market...</td>
</tr>
<tr>
<td>DELALL</td>
<td>3</td>
<td>Delete all events from the cache. This is a market depth flush usually passed at the start or end of trading or when a trading halt occurs.</td>
</tr>
<tr>
<td>DELBETTER</td>
<td>4</td>
<td>Delete this order and any superior orders. The order ID at the next inferior position is now the best order. This differs from the EXEC command...</td>
</tr>
<tr>
<td>DELSIDE</td>
<td>5</td>
<td>Delete all events on the corresponding side (bid/ask) of the order book.</td>
</tr>
<tr>
<td>EXEC</td>
<td>7</td>
<td>Trade Execution. Find the corresponding order in the cache, replace event details with this event and then delete any prior superior orders.</td>
</tr>
<tr>
<td>MOD</td>
<td>8</td>
<td>Modify an existing event in the market depth cache. Find the cached market depth event by the position in the new market depth event and replace...</td>
</tr>
<tr>
<td>REPLACE</td>
<td>10</td>
<td>Replace previous price level or order at this position. Add price level or order if you do not have it currently in the cache. A zero (0) price...</td>
</tr>
</tbody>
</table>
The other important enumeration value is found in the Book Type (MD_BOOK_TYPE) field and is only included in the initial paint message. Here is a complete table covers all three book types and their possible table command enumeration values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLACE_BY_BROKER</td>
<td>11</td>
<td>This table command is used for top of file feeds where the action is to replace by the broker mnemonic. The recipient needs to find the broker in their cache and replace the quote with the one in the market depth event. If that broker is not present, it should be added to the cache. If the price and size for a broker is set to 0, the broker should be deleted from the cache.</td>
</tr>
<tr>
<td>CLEARALL</td>
<td>12</td>
<td>Clears the entire orderbook for the specified side. This market depth table command is issued by Bloomberg when market depth recovery is occurring. This table command has the same effect on the cache as DELETEALL which means all order or levels should be cleared from the cache. During LVC recovery you will generally see 2 CLEARALLs - 1 for Bid side and 1 for Ask side. Should the client of market depth need to process a recovery of market depth differently, this table command allows the user to differentiate from the source/exchange produced DELETEALL. CLEARALL messages may occur without accompanying RETRANS labels in the event of data loss within Bloomberg network or upon the receipt of the first tick of a new trading day. Hence, upon the receipt of a CLEARALL, you should clear your book and prepare to receive the subsequent recover ADD messages.</td>
</tr>
<tr>
<td>REPLACE_CLEAR</td>
<td>13</td>
<td>The REPLACE_CLEAR table command is intended to remove an order or more often a level in the market depth cache. The REPLACE_CLEAR should be indexed by the MarketDepth.ByLevel/ByOrder.Bid/Ask.Position field. The cache should NOT be shifted up after the level is cleared. A clear means all orders at that position have been deleted from the order book. It is possible that an order or level at a superior or most superior position to be cleared prior to more inferior levels. After the level is cleared in this case, it is expected that subsequent market depth event(s) will be passed to clear the orders or levels at positions inferior to the one just cleared.</td>
</tr>
</tbody>
</table>

The other important enumeration value is found in the Book Type (MD_BOOK_TYPE) field and is only included in the initial paint message. Here is a complete table covers all three book types and their possible table command enumeration values.
The following code snippet demonstrates how to handle and print out a MarketDepth subscription to std::cout. This C++ snippet is based on the aforementioned "MarketDepthSubscriptionExample" C++ SDK example. For a more complete example that demonstrates how to handle and build an order/level book, please reference the aforementioned "MarketDepthSubscriptionSnapshotExample" example in either the Java, C++ or .NET SDK.

```cpp
bool processEvent(const Event &event, Session *session) {
  try {
    switch (event.eventType()) {
      case Event::SUBSCRIPTION_DATA:
        {
          char timeBuffer[64];
          getTimeStamp(timeBuffer, sizeof(timeBuffer));
          std::cout << "Processing SUBSCRIPTION_DATA" << std::endl;
          MessageIterator msgIter(event);
          while (msgIter.next()) {
            Message msg = msgIter.message();
            std::string *topic = reinterpret_cast<std::string*>(
                                  msg.correlationId().asPointer());
            std::cout << timeBuffer << " : " << topic->c_str() << " - ";
            msg.print(std::cout);
          }
          break;
        }
      case Event::SUBSCRIPTION_STATUS:
        return processSubscriptionStatus(event);
      break;
      default:
        return processMiscEvents(event);
        break;
    }
    catch (Exception &e) {
      std::cout << "Library Exception !!! " << e.description().c_str() << std::endl;
    }
    return false;
  }
}
```

**Figure 9-2: Handling a market depth data update (C++)**
You will notice that the above code checks the EventType being returned and looks for SUBSCRIPTION_DATA. Please note that the processSubscriptionStatus() and processMiscEvents() functions were not shown for brevity. You will also notice that the event handler for the tick updates is identical to that of a //blp/mktdata subscription, for instance.

Handling Multiple Messages (a.k.a. Fragments)

The summary (initial paint) messages can be split into one or more smaller messages in the case where the returned data is too large to fit into a single message. It will be up to you to handle this in your application.

You will achieve this by checking the Fragment type of any SUBSCRIPTION_DATA event message containing a MKTDEPTH_EVENT_SUBTYPE of value "TABLE_INITPAINT". The Fragment enum is used to indicate whether a message is a fragmented message or not and what position it occurs within the chain of split fragmented messages. If the TABLE_INITPAINT is split into two parts, then the first message will have a Fragment type value of FRAGMENT_START and a last message of FRAGMENT_END. If the TABLE_INITPAINT is split into more than 2 parts, all middle Fragments will be of type FRAGMENT_INTERMEDIATE.

This enum will exist in both MARKET_BY_ORDER and MARKET_BY_LEVEL messages.

<table>
<thead>
<tr>
<th>Message::Fragment Type Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGMENT_NONE</td>
<td>Message is not fragmented</td>
</tr>
<tr>
<td>FRAGMENT_START</td>
<td>The first fragmented message</td>
</tr>
<tr>
<td>FRAGMENT_INTERMEDIATE</td>
<td>Intermediate fragmented messages</td>
</tr>
<tr>
<td>FRAGMENT_END</td>
<td>The last fragmented message</td>
</tr>
</tbody>
</table>

The following code snippet demonstrates how the C++ "MarketDepthSubscriptionSnapshotExample" example checks the fragment type. Please take a look at the full code example in the SDK for a working version of this code.

```cpp
if (subType == TABLE_INITPAINT) {
    if (msg.fragmentType() == BloombergLP::blpapi::Message::Fragment::FRAGMENT_START ||
        msg.fragmentType() == BloombergLP::blpapi::Message::Fragment::FRAGMENT_NONE) {
        if (msg.hasElement(MBO_WINDOW_SIZE, true))
            d_orderBooks[Side::ASKSIDE].window_size = (unsigned int)
                msg.getElementAsInt64(MBO_WINDOW_SIZE);
        d_orderBooks[Side::BIDSIDE].window_size =
            d_orderBooks[Side::ASKSIDE].window_size;
    }
    d_orderBooks[Side::ASKSIDE].book_type = msg.getElementAsString(MD_BOOK_TYPE);
    // clear cache
    d_orderBooks[Side::ASKSIDE].doClearAll();
    d_orderBooks[Side::BIDSIDE].doClearAll();
}
```
The above code checks the Market Depth Event Sub-Type being returned, and if it equals TABLE_INITPAINT, then it checks the Fragment Type. If a FRAGMENT_START or FRAGMENT_NONE type is returned by msg.fragmentType(), then the order book is cleared.

**Data Response for ADD-MOD-DEL (AMD) Order Books**

Every event in an Add-Mode-Delete (AMD) order book is critical in maintaining an accurate book. One missed message can result in a book that is wrong for the remainder of the trading day. Because of this, all AMD market depth messages have a MBO_SEQNUM_RT field with a non-zero value. This field is generated by the Bloomberg ticker plant when it creates its order book and increments monotonically for every update. Separate counters are maintained for the bid and ask sides since they update independently.

It is up to your application to clear the book as soon as you receive an initial paint message.
MBO-AMD sample subscription output (for "/blp/mktdepth/bsym/CT/RIM?type=MBO")

```
Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
    MKTDEPTH_EVENT_SUBTYPE = TABLE_INITPAINT
    ID_BB_SEC_NUM_SRC = 502511690826
    FEED_SOURCE = "CT"
    EID = 14184
    MD_TABLE_CMD_RT = ADD
    MD_BOOK_TYPE = MBO-AMD
    MBO_WINDOW_SIZE = 200
    MBL_TABLE_ASK[] = {
    }
    MBL_TABLE_BID[] = {
    }
    MBO_TABLE_ASK[] = {
        MBO_TABLE_ASK = {
            MBO_ASK_POSITION_RT = 1
            MBO_ASK_RT = 11.3199996948242
            MBO_ASK_BROKER_RT = "   1"
            MBO_ASK_COND_CODE_RT = ""
            MBO_ORDER_ID_RT = "3235323500004c1d0001"
            MBO_ASK_SIZE_RT = 200
            MBO_TIME_RT = 2012-05-25T19:53:06.000+00:00
            MD_TABLE_CMD_RT = ADD
        }
    }
    MBO_TABLE_ASK = {
        MBO_ASK_POSITION_RT = 2
        MBO_ASK_RT = 11.3199996948242
        MBO_ASK_BROKER_RT = "   1"
        MBO_ASK_COND_CODE_RT = ""
        MBO_ORDER_ID_RT = "3235323500004c1e0001"
        MBO_ASK_SIZE_RT = 100
        MBO_TIME_RT = 2012-05-25T19:53:06.000+00:00
        MD_TABLE_CMD_RT = ADD
    }
    ... (more)
```
MBO_TABLE_BID[] = {
    MBO_TABLE_BID = {
        MBO_BID_POSITION_RT = 1
        MBO_BID_RT = 11.3100004196167
        MBO_BID_BROKER_RT = "79"
        MBO_BID_COND_CODE_RT = ""
        MBO_ORDER_ID_RT = "32353235000075f8004f"
        MBO_BID_SIZE_RT = 1400
        MBO_TIME_RT = 2012-05-25T19:46:59.000+00:00
        MD_TABLE_CMD_RT = ADD
    }
    MBO_TABLE_BID = {
        MBO_BIDPOSITION_RT = 2
        MBO_BID_RT = 11.3100004196167
        MBO_BID_BROKER_RT = "79"
        MBO_BID_COND_CODE_RT = ""
        MBO_ORDER_ID_RT = "323532350000761a004f"
        MBO_BID_SIZE_RT = 500
        MBO_TIME_RT = 2012-05-25T19:47:33.000+00:00
        MD_TABLE_CMD_RT = ADD
    }
    ... (more)
}

Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
    MKTDEPTH_EVENT_SUBTYPE = ASK
    EID = 14184
    MD_TABLE_CMD_RT = DEL
    MBO_SEQNUM_RT = 199951
    MBO_ASK_POSITION_RT = 7
    MBO_ASK_RT = 11.3199996948242
    MBO_ASK_BROKER_RT = "79"
    MBO_ASK_COND_CODE_RT = ""
    MBO_ORDER_ID_RT = "323532350000774e004f"
    MBO_ASK_SIZE_RT = 500
    MBO_TIME_RT = 2012-05-25T19:47:33.000+00:00
    MBL_TABLE_ASK[] = {
    }
    MBL_TABLE_BID[] = {
    }
    MBO_TABLE_ASK[] = {
    }
    MBO_TABLE_BID[] = {
    }
}
Processing SUBSCRIPTION_DATA
/bsym/CT/RIM - MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
    MKTDEPTH_EVENT_SUBTYPE = TABLE_INITPAINT
    ID_BB_SEC_NUM_SRC = 502511690826
    FEED_SOURCE = "CT"
    EID = 14184
    MD_TABLE_CMD_RT = ADD
    MD_BOOK_TYPE = MBO-AMD
    MBO_WINDOW_SIZE = 200
    MBL_TABLE_ASK[] = {
        }
    MBL_TABLE_BID[] = {
        }
    MBO_TABLE_ASK[] = {
        MBO_TABLE_ASK = {
            MBO_ASK_POSITION_RT = 200
            MBO_ASK_RT = 12
            MBO_ASK_BROKER_RT = "80"
            MBO_ASK_COND_CODE_RT = ""
            MBO_ORDER_ID_RT = "3235313500000c390050"
            MBO_ASK_SIZE_RT = 100
            MBO_TIME_RT = 2012-05-25T15:20:49.000+00:00
            MD_TABLE_CMD_RT = ADD
        }
    }
    MBO_TABLE_BID[] = {
        }
}

Notes:

The first message above is the initial paint (as indicated by the TABLE_INITPAINT event sub-type (i.e., MKTDEPTH_EVENT_SUBTYPE)) and indicates that it is a Market-By-Order message, as indicated by the MARKET_BY_ORDER event type (i.e., MKTDEPTH_EVENT_TYPE). Within the initial paint message, you will find a table of asks and bids. In this case, it is an MBO request, so the table will be of MBO bids and asks (indicated by MBO_TABLE_BID[] and MBO_TABLE_ASK[] array items). When you receive an initial paint message, you should clear your book prior to populating with the table of Asks and Bids.

Because this is an AMD (Add-Mod-Del) MBO Book Type, the MD_TABLE_CMD_RT field in the initial paint is ADD. The valid table commands for subsequent AMD type message updates are ADD, MOD, DELETE and CLEARALL.
Because the Replace-By-Broker (RBB) method addresses individual broker orders, it applies only to MBO order books. Unlike AMD and RBL, there is no concept of row numbers in an RBB order book. Instead each broker ID represents a row. This leaves it up to the feed handler to decide how to order the book. Typically they are ordered by best (highest) bid and best (lowest) ask to worst (lowest) bid and worst (highest) ask. If multiple orders exist at the same price on the same side then they can be sorted by size or by broker code. It is up to your application to clear the book as soon as you receive an initial paint message.

**MBO-RBB Subscription Output** (for "/blp/mktdepth/bsym/US/AAPL?type=MBO")

```plaintext
Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
    MKTDEPTH_EVENT_SUBTYPE = TABLE_INITPAINT
    ID_BB_SEC_NUM_SRC = 399432471918
    FEED_SOURCE = "US"
    EID = 14023
    MD_TABLE_CMD_RT = REPLACE_BY_BROKER
    MD_BOOK_TYPE = MBO-RBB
    MBL_TABLE_ASK[] = {
    }
    MBL_TABLE_BID[] = {
    }
    MBO_TABLE_ASK[] = {
        MBO_TABLE_ASK = {
            MBO_ASK_RT = 604.630126953125
            MBO_ASK_BROKER_RT = "ADAM"
            MBO_ASK_BROKER_MODE_RT = OPEN
            MBO_ASK_COND_CODE_RT = ""
            MBO_ASK_COND_CODE_SRC_RT = ""
            MBO_ASK_LSRC_RT = "UQ"
            MBO_ASK_SIZE_RT = 100
            MBO_TIME_RT = 2012-05-25T13:44:01.000+00:00
            MD_TABLE_CMD_RT = REPLACE_BY_BROKER
        }
        MBO_TABLE_ASK = {
            MBO_ASK_RT = 560.75
            MBO_ASK_BROKER_RT = "ARCX"
            MBO_ASK_BROKER_MODE_RT = OPEN
            MBO_ASK_COND_CODE_RT = ""
            MBO_ASK_COND_CODE_SRC_RT = ""
            MBO_ASK_LSRC_RT = "UP"
            MBO_ASK_SIZE_RT = 200
            MBO_TIME_RT = 2012-05-25T19:24:12.000+00:00
            MD_TABLE_CMD_RT = REPLACE_BY_BROKER
        }
    ... (more)
}
```
MBO_TABLE_BID[] = {
    MBO_TABLE_BID = {
        MBO_BID_RT = 514.900146484375
        MBO_BID_BROKER_RT = "ADAM"
        MBO_BID_BROKER_MODE_RT = OPEN
        MBO_BID_COND_CODE_RT = ""
        MBO_BID_COND_CODE_SRC_RT = ""
        MBO_BID_LSRC_RT = "UQ"
        MBO_BID_SIZE_RT = 100
        MBO_TIME_RT = 2012-05-25T13:44:01.000+00:00
        MD_TABLE_CMD_RT = REPLACE_BY_BROKER
    }
    MBO_TABLE_BID = {
        MBO_BID_RT = 560.60009765625
        MBO_BID_BROKER_RT = "ARCX"
        MBO_BID_BROKER_MODE_RT = OPEN
        MBO_BID_COND_CODE_RT = ""
        MBO_BID_COND_CODE_SRC_RT = ""
        MBO_BID_LSRC_RT = "UP"
        MBO_BID_SIZE_RT = 200
        MBO_TIME_RT = 2012-05-25T19:24:13.000+00:00
        MD_TABLE_CMD_RT = REPLACE_BY_BROKER
    }
    ... (more)
}

Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
    MKTDEPTH_EVENT_SUBTYPE = BID
    EID = 14023
    MD_TABLE_CMD_RT = REPLACE_BY_BROKER
    MBO_TIME_RT = 2012-05-25T19:24:14.000+00:00
    MBO_BID_RT = 560.56005859375
    MBO_BID_BROKER_RT = "NQBX"
    MBO_BID_BROKER_MODE_RT = OPEN
    MBO_BID_COND_CODE_RT = ""
    MBO_BID_COND_CODE_SRC_RT = ""
    MBO_BID_LSRC_RT = "UB"
    MBO_BID_SIZE_RT = 100
    MBL_TABLE_ASK[] = {
    }
    MBL_TABLE_BID[] = {
    }
    MBO_TABLE_ASK[] = {
    }
    MBO_TABLE_BID[] = {
    }
}
Notes:

The first message above is the initial paint (as indicated by the TABLE_INITPAINT event sub-type (i.e., MKTDEPTH_EVENT_SUBTYPE)) and indicates that it is a Market-By-Order message, as indicated by the MARKET_BY_ORDER event type (i.e., MKTDEPTH_EVENT_TYPE). Within the initial paint message, you will find a table of asks and bids. In this case, it is an MBO request, so the table will consist of MBO bids and asks (indicated by MBO_TABLE_BID[] and MBO_TABLE_ASK[] array items). When you receive an initial paint message, you should clear your book prior to populating with the array of Asks and Bids.

Because this is a Request-By-Broker (RBB) MBO Book Type, the MD_TABLE_CMD_RT field in the initial paint and subsequent update is REPLACE_BY_BROKER. The other valid table commands for an RBB type are REPLACE_CLEAR and CLEARALL, which are sent by the exchange.

Data Response For Request-By-Level (RBL) Order Books

With the Replace-By-Level (RBL) method each level is explicitly sent so that to maintain the order book the feed handler simply has to apply the data for each level directly. There is no shifting of rows in the order book. Because each level is maintained individually (unlike the AMD method) missed messages, while never a good thing, have no impact other than that they were missed. All other levels retain their correct values.

The RBL method is generally easier to implement than AMD, but this comes with a cost. Because each level is maintained individually a new value at level one requires that the entire

Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
  MKTDEPTH_EVENT_TYPE = MARKET_BY_ORDER
  MKTDEPTH_EVENT_SUBTYPE = BID
  EID = 14023
  MD_TABLE_CMD_RT = REPLACE_BY_BROKER
  MBO_TIME_RT = 2012-05-25T19:24:14.000+00:00
  MBO_BID_RT = 560.60009765625
  MBO_BID_BROKER_RT = "ARCX"
  MBO_BID_BROKER_MODE_RT = OPEN
  MBO_BID_COND_CODE_RT = ""
  MBO_BID_COND_CODE_SRC_RT = ""
  MBO_BID_LSRC_RT = "UP"
  MBO_BID_SIZE_RT = 100
  MBL_TABLE_ASK[] = {
  }
  MBL_TABLE_BID[] = {
  }
  MBO_TABLE_ASK[] = {
  }
  MBO_TABLE_BID[] = {
  }
}


order book be resent. The bandwidth impact for small order books is minimal but can be extreme for large order books. For this reason AMD is often used for large order books.

**MBL-RBL Subscription Output** (for “//blp/mktdepth/ticker/ESM2 Index?type=MBL”),

```
Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_LEVEL
    MKTDEPTH_EVENT_SUBTYPE = TABLE_INITPAINT
    ID_BB_SEC_NUM_SRC = 2078784978839
    FEED_SOURCE = "eCME"
    EID = 14002
    MD_TABLE_CMD_RT = REPLACE
    MD_BOOK_TYPE = MBL-RBL
    MBL_WINDOW_SIZE = 10
    MBL_TABLE_ASK[] = {
        MBL_TABLE_ASK = {
            MBL_ASK_POSITION_RT = 1
            MBL_ASK_RT = 1314.75
            MBL_ASK_COND_CODE_RT = 
            MBL_ASK_NUM_ORDERS_RT = 35
            MBL_ASK_SIZE_RT = 384
            MBL_TIME_RT = 2012-05-25T20:05:13.302+00:00
            MD_TABLE_CMD_RT = REPLACE
        }
        MBL_TABLE_ASK = {
            MBL_ASK_POSITION_RT = 2
            MBL_ASK_RT = 1315
            MBL_ASK_COND_CODE_RT = 
            MBL_ASK_NUM_ORDERS_RT = 65
            MBL_ASK_SIZE_RT = 397
            MBL_TIME_RT = 2012-05-25T20:05:13.648+00:00
            MD_TABLE_CMD_RT = REPLACE
        }
    ...
    ... (more)
```
MBL_TABLE_BID[] = {
    MBL_TABLE_BID = {
        MBL_BID_POSITION_RT = 1
        MBL_BID_RT = 1314.5
        MBL_BID_COND_CODE_RT = ""
        MBL_BID_NUM_ORDERS_RT = 65
        MBL_TIME_RT = 2012-05-25T20:05:13.043+00:00
        MBL_BID_SIZE_RT = 427
        MD_TABLE_CMD_RT = REPLACE
    }
    MBL_TABLE_BID = {
        MBL_BID_POSITION_RT = 2
        MBL_BID_RT = 1314.25
        MBL_BID_COND_CODE_RT = ""
        MBL_BID_NUM_ORDERS_RT = 69
        MBL_TIME_RT = 2012-05-25T20:05:11.351+00:00
        MBL_BID_SIZE_RT = 631
        MD_TABLE_CMD_RT = REPLACE
    }
    ... (more)
}

Processing SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_LEVEL
    MKTDEPTH_EVENT_SUBTYPE = ASK
    EID = 14002
    MD_TABLE_CMD_RT = REPLACE
    MD_MULTI_TICK_UPD_RT = 0
    MBL_ASK_POSITION_RT = 2
    MBL_ASK_RT = 1315
    MBL_ASK_COND_CODE_RT = ""
    MBL_ASK_NUM_ORDERS_RT = 66
    MBL_ASK_SIZE_RT = 398
    MBL_TIME_RT = 2012-05-25T20:05:14.085+00:00
    MBL_TABLE_ASK[] = {
    }
    MBL_TABLE_BID[] = {
    }
    MBO_TABLE_ASK[] = {
    }
    MBO_TABLE_BID[] = {
    }
}


Processing_SUBSCRIPTION_DATA
MarketDepthUpdates = {
    MKTDEPTH_EVENT_TYPE = MARKET_BY_LEVEL
    MKTDEPTH_EVENT_SUBTYPE = ASK
    EID = 14002
    MD_TABLE_CMD_RT = REPLACE
    MD_MULTI_TICK_UPD_RT = 0
    MBL_ASK_POSITION_RT = 2
    MBL_ASK_RT = 1315
    MBL_ASK_COND_CODE_RT = 
    MBL_ASK_NUM_ORDERS_RT = 65
    MBL_ASK_SIZE_RT = 397
    MBL_TIME_RT = 2012-05-25T20:05:14.148+00:00
    MBL_TABLE_ASK[] = {
        }
    MBL_TABLE_BID[] = {
        }
    MBO_TABLE_ASK[] = {
        }
    MBO_TABLE_BID[] = {
        }
}

Notes:
The first message above is the initial paint (as indicated by the TABLE_INITPAINT event sub-type (i.e. MKTDEPTH_EVENT_SUBTYPE)) and indicates that it is a Market-By-Level (MBL) message, as indicated by the MARKET_BY_LEVEL event type (i.e. MKTDEPTH_EVENT_TYPE). Within the initial paint message, you will find the MBL_WINDOW_SIZE. This indicates the number of levels in the book, along with the table command (i.e. MD_TABLE_CMD_RT) with a value of "REPLACE" and book type (i.e. MD_BOOK_TYPE) with a value of "MBL-RBL".

Because this is a Request-By-Level (RBL) MBL Book Type, the MD_TABLE_CMD_RT field in the initial paint is REPLACE and all subsequent updates will possess a table command of either REPLACE_CLEAR, REPLACE or CLEARALL. This is true for both MBO and MBL event types. The output above includes a sample BID/REPLACE and ASK/REPLACE_CLEAR message.

Order Book Recaps
Order book recaps provide all the information required to completely rebuild an order book. They can be initiated by the exchange, B-Pipe or the client application.

Recaps apply to every style of order book: Add-Mod-Delete (AMD), Replace-by-Level (RBL) and Replace-by-Broker (RBB), but they play a special role for AMD order books. It is critical that AMD order books receive every message. A single missed message (a data gap) can result in the AMD book being wrong for the remainder of the market day. RBL and RBB books tend to be self-correcting in the event of a data gap making gap detection less critical.
The MBL_SEQNUM_RT and MBL_SEQNUM_RT fields are sequentially increasing numbers included only in AMD order book market depth messages. They allow the client application to detect gaps in the AMD market depth messages. A sequence number 5 followed by 7 indicates that a gap of one message occurred.

**Gap Detection**

Data gaps occur as a result of missed network messages. While rare, as in every complex networked system, missed messages can occur at any level and for many reasons. If a data gap occurs between the B-Pipe order book systems and the application, it is the client application's responsibility to take action to restore the order book to an accurate state. If the gap is detected by the Bloomberg upstream order book systems, B-Pipe will automatically initiate the recap without any action by the client application.

When B-Pipe detects a gap in the MBL or MBO "AMD" order book, the MD_GAP_DETECTED field is present and set to "true" in every market depth update message for each effected order book. This informs the client application that B-Pipe has detected the gap and to expect an automatic recap.

MD_GAP_DETECTED will not be present once the recap is sent. Therefore, even though a client application detects a gap, if this field is present in market depth update messages, no further action is required by the client application except to begin reading the recap messages, which will follow immediately and be indicated with a MKTDEPTH_EVENT_SUBTYPE of BID_RETRANS and ASK_RETRANS in each message update. In cases where a sequence number gap is detected but the MD_GAP_DETECTED field is not present in the message, it is the responsibility of the client application to request a recap (i.e., resubscribe) to the order book.

<table>
<thead>
<tr>
<th>Fields</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKTDEPTH_EVENT_SUBTYPE</td>
<td>Present in every market depth message for all styles of orderbook. When an unsolicited recap is in progress, this field will have a value of &quot;BID_RETRANS&quot; or &quot;ASK_RETRANS&quot;.</td>
</tr>
<tr>
<td>MBL_SEQNUM_RT and MBL_SEQNUM_RT</td>
<td>Present in every market depth message for AMD, and only AMD, order books. They will have a value of 0 if the message is part of an order book recap, regardless of how initiated. Gap detection does not apply to recaps. The value of these fields in the first non-recap market depth update message following the recap will have a non-zero value which should be used to detect any gaps following the recap.</td>
</tr>
<tr>
<td>MD_TABLE_CMD_RT</td>
<td>Present in every market depth message, it indicates the action to take for this market depth message. The behavior of this field is unchanged. A value of &quot;DELSIDE&quot; indicates that the appropriate side of the order book (bid or ask) should be cleared of all values. All recaps start with a DELSIDE. All other values should be applied as already documented.</td>
</tr>
</tbody>
</table>
Table 9-1: Fields Affected by Recaps

<table>
<thead>
<tr>
<th>Fields</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_MULTI_TICK_UPD_RT</td>
<td>When present, indicates that a market depth message is one of multiple messages that make up a single update to an order book. A value of 1 indicates that additional market depth messages that are part of the same order book update will follow this message. A value of 0 indicates that this is the last message in the update and that the update is complete. All recaps for every style of order book are sent as multi-tick updates. Multi-tick updates may also be used to send non-recap RBL style order book updates.</td>
</tr>
</tbody>
</table>

Frequently Asked Question:

1. For a book with a book size of 5 and 5 active levels, what happens when the exchange needs to delete level 3?

   The answer varies based on the book type. For instance,

   For BookType=MB[LO]-RBL:
   REPLACE POSITION=3
   REPLACE POSITION=4
   REPLACE_CLEAR POSITION=5

   For Booktype=MB[LO]-AMD:
   DELETE POSITION=3

9.2.2 Market List Service

Overview

The Market List Service (//blp/mktlist) is used to perform two types of list data operations. The first is to subscribe to lists of instruments, known as chains, using the 'chain' <subservice name> (i.e. //blp/mktlist/chain). The second is to request a snapshot list of all the instruments that match a given topic key using the 'secids' <subservice name> (i.e. //blp/mktlist/secids). The //blp/mktlist service is available to both BPS (Bloomberg Professional Service) and Non-BPS users.

The syntax of the Market List subscription string is as follows:

    //<service owner>/<service name>/<subservice name>/<topic>

where <topic> is comprised of '<topic type>/<topic key>' and <subservice name> is either 'chain' or 'secids'. Table 9-2 below provides further details.
Table 9-2: Market List String Definitions

<table>
<thead>
<tr>
<th>&lt;service owner&gt;</th>
<th>For B-Pipe is “blp”</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;service name&gt;</td>
<td>For subscription and snapshot data is “mktlist”</td>
</tr>
<tr>
<td>&lt;subservice name&gt;</td>
<td>/chain Subscription-based request for a list of instruments. It can be one of a variety of types such as &quot;Option Chains&quot;, &quot;Index Members&quot;, &quot;EID List&quot;, &quot;GDCO List&quot; or &quot;Yield Curve&quot;. See Table 9-4 below for additional information and examples of each.</td>
</tr>
<tr>
<td></td>
<td>/secids Snapshot request for one-time list of instruments that match a given &lt;topic&gt;. It will always be &quot;Secids List&quot;. See Table 9-4 below for additional information and an example.</td>
</tr>
<tr>
<td>&lt;topic type&gt;</td>
<td>/cusip Requests by CUSIP</td>
</tr>
<tr>
<td></td>
<td>/sedol Requests by SEDOL</td>
</tr>
<tr>
<td></td>
<td>/isin Requests by ISIN</td>
</tr>
<tr>
<td></td>
<td>/bsid Requests by Bloomberg Security Identifier</td>
</tr>
<tr>
<td></td>
<td>/bsym For requests by Bloomberg Security Symbol</td>
</tr>
<tr>
<td></td>
<td>/buid For requests by Bloomberg Unique Identifier</td>
</tr>
<tr>
<td></td>
<td>/eid For requests by Entitlement ID</td>
</tr>
<tr>
<td></td>
<td>/source For requests by Source syntax</td>
</tr>
<tr>
<td></td>
<td>/gdco For Requests by GDCO syntax</td>
</tr>
<tr>
<td></td>
<td>/bpkbl Requests by Bloomberg parsekeyable Identifier</td>
</tr>
<tr>
<td></td>
<td>/esym Requests by Exchange Symbol</td>
</tr>
<tr>
<td></td>
<td>/ticker Requests by Bloomberg ticker</td>
</tr>
<tr>
<td></td>
<td>/bbgid Requests by Bloomberg Global Identifier</td>
</tr>
<tr>
<td>&lt;topic key&gt;a</td>
<td>The following topic types consist of source and the value of a given identifier separated by the forward slash. &lt;source&gt;/&lt;identifier&gt;</td>
</tr>
<tr>
<td></td>
<td>/cusip</td>
</tr>
<tr>
<td></td>
<td>/sedol</td>
</tr>
<tr>
<td></td>
<td>/isin</td>
</tr>
<tr>
<td></td>
<td>/bpkbl</td>
</tr>
<tr>
<td></td>
<td>/buid</td>
</tr>
<tr>
<td></td>
<td>/bsym</td>
</tr>
<tr>
<td></td>
<td>/bbgid</td>
</tr>
<tr>
<td></td>
<td>The following topic types do not require a source and consist of value alone &lt;Identity&gt;</td>
</tr>
<tr>
<td></td>
<td>/bsid</td>
</tr>
<tr>
<td></td>
<td>/eid</td>
</tr>
<tr>
<td></td>
<td>/ticker</td>
</tr>
<tr>
<td></td>
<td>The following topic type consists of only a &lt;source&gt; &lt;broker_id&gt;/&lt;mon_id&gt;</td>
</tr>
<tr>
<td></td>
<td>/source</td>
</tr>
<tr>
<td></td>
<td>/gdco</td>
</tr>
</tbody>
</table>

a. See examples in Table 9-4 on page 142.
You will find two separate examples in the B-Pipe SDK for C++, Java, and .NET. They are as follows:

- **MarketListSubscriptionExample**
  This example demonstrates how to make a simple Market List "chain" subscription for one, or more, securities and displays all of the messages to the console window.

- **MarketListSnapshotExample**
  This example demonstrates how to make a Market List "secids" snapshot request and displays the message to the console window.

Now that you have a better understanding as to how a //blp/mktlist subscription or snapshot string is formed, it is now time to use it in your application. The following sections provide further details as to how to subscribe to a chain of instruments and request a Snapshot of a list of members.

### Subscribing To Instrument Chains

#### Overview

B-Pipe supports the ability to subscribe to lists of instruments known as chains. When a subscription is made for a chain, the request must first resolve to a single B-Pipe instrument. This instrument is called the “underlying instrument”.

The instruments returned in the list are referred to as "list members". The characteristics of list members depends upon the security class of the underlying instrument or parameters included in the initial chain request. Examples are list members that are options or members that are futures.

In most cases, the list members will all be the same security class. When the underlying security class is an Index or Curve, the security class of the each member may or may not be same.

In most cases, underlying instruments are regular B-Pipe instruments, such as an equity or futures contract. Other times, the underlying instrument will be a pseudo instrument whose sole purpose is to serve as the underlying instrument for the chain. Like all other instruments on B-Pipe, the underlying pseudo-instrument has its own, unique ID_BB_SEC_NUM_SRC. It can be subscribed to as a regular instrument but since it has no price data of its own the subscription will only return reference data.

For most chains, the relationship between the underlying instrument and the list members is established by the B-Pipe service when the subscription is made using the BSID of the underlying instrument. Every member of the list has a LIST_UNDERLYING_ID_BSID field, which contains the BSID value of the underlying instrument, and all matching instruments of the appropriate security class are returned in the list of members.
Index and Curve lists are handled differently. The list's members are maintained by the Bloomberg Data Center. Once it is determined that this list subscription is for index or curve members, the Bloomberg Data Center is queried for the list of members. This list contains the terminal ticker (ParseKeyable symbol) for each member, which is resolved to an instrument on B-Pipe. It is possible that an index or curve list member is not available on B-Pipe. In this case, the list member will be included in the list, but return only the ParseKeyable symbol. This allows the requestor to contact Bloomberg about getting the missing instrument added to B-Pipe.

The default security class of the list members depends on the security class of the underlying instrument specified in the request. The default can be overridden using the optional parameter "secclass". Table 9-3 defines the default security class of the list members for each underlying instrument security class.

<table>
<thead>
<tr>
<th>Underlying Security Class</th>
<th>Default Chain Member Security Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>Option</td>
</tr>
<tr>
<td>Equity</td>
<td>Option</td>
</tr>
<tr>
<td>Fixed Income</td>
<td>N/A</td>
</tr>
<tr>
<td>Fund</td>
<td>Option</td>
</tr>
<tr>
<td>Future Root</td>
<td>Future</td>
</tr>
<tr>
<td>Future Contract</td>
<td>Option</td>
</tr>
<tr>
<td>Index</td>
<td>Members</td>
</tr>
<tr>
<td>Option</td>
<td>N/A</td>
</tr>
<tr>
<td>Warrant</td>
<td>N/A</td>
</tr>
<tr>
<td>Curve</td>
<td>Members</td>
</tr>
</tbody>
</table>

An alternate security class for the returned members is available and can be specified in the subscription string using a parameter. For example, the following chain requests are equivalent because the default member security class is Option:

//blp/mktlist/chain/bsym/US/IBM
//blp/mktlist/chain/bsym/US/IBM;secclass=Option

However, by using a parameter, we can obtain a list of Futures with IBM as the underlying instrument:

//blp/mktlist/chain/bsym/US/IBM;secclass=Future

In order to further qualify the subscription string, a parameter "source" can be applied. The value of this parameter is assigned by the user or application to limit the amount of returned members to those belonging to the specified source(s) only. More than one value is allowed for this parameter.
The "source" can be substituted by a "~". This value can be used when the client assumes that there is only one source for the security and there is no actual need to specify it. If this is the case, the subscription request will be processed successfully, but if the security has more than one source and the request is ambiguous, then the client will receive a SubscriptionFailure response with a NOTUNIQUE description. An example of such a subscription string would be "/blp/mktlist/chain/cusip/~459200101".

Table 9-4: Chain Subservice Examples

<table>
<thead>
<tr>
<th>Type of Chain List</th>
<th>Example Subscription String</th>
<th>Topic Type</th>
<th>Topic Keya</th>
<th>Refreshesb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option Chains</td>
<td>//blp/mktlist/chain/bsym/LN/VOD</td>
<td>/bsym</td>
<td>&lt;DX282&gt;/&lt;DY003&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/bsid/6768605358297</td>
<td>/bsid</td>
<td>&lt;ID122&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/buid/LN/EQ0010160500001000</td>
<td>/buid</td>
<td>&lt;DX282&gt;/&lt;ID059&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/bbid/LN/EQ0010160500001000</td>
<td>/bbid</td>
<td>&lt;DX262&gt;/&lt;ID059&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/bpkbl/VOD LN Equity</td>
<td>/bpkbl</td>
<td>&lt;DX194&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/esym/LN/VOD</td>
<td>/esym</td>
<td>&lt;DX282&gt;/&lt;EX005&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/cusip/UN/459200101</td>
<td>/cusip</td>
<td>&lt;DX282&gt;/&lt;ID032&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/isin/LN/GB00B4H4HKS39</td>
<td>/isin</td>
<td>&lt;DX282&gt;/&lt;ID005&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/sedol/LN/BH4HKS3</td>
<td>/sedol</td>
<td>&lt;DX282&gt;/&lt;ID002&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/bbgid/LN/BBG000C6K6G9</td>
<td>/bbgid</td>
<td>&lt;DX282&gt;/&lt;ID135&gt;</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/chain/ticker/VOD LN Equity</td>
<td>/ticker</td>
<td>&lt;DX194&gt;</td>
<td>No</td>
</tr>
<tr>
<td>Index List</td>
<td>//blp/mktlist/chain/bsym/FTUK/UKX Index;secclass=Option</td>
<td>/bsym</td>
<td>&lt;DX282&gt;/&lt;DY003&gt;</td>
<td>Daily</td>
</tr>
<tr>
<td>Yield Curve</td>
<td>//blp/mktlist/chain/bpkbl/YCMM0010 Index</td>
<td>/bpkbl</td>
<td>&lt;identifier&gt;</td>
<td>Daily</td>
</tr>
<tr>
<td>GDCO</td>
<td>/blp/mktlist/chain/gdco/broker/id</td>
<td>/gdco</td>
<td>&lt;broker_id&gt;/&lt;mon_id&gt;</td>
<td>N/A</td>
</tr>
<tr>
<td>EID List</td>
<td>/blp/mktlist/chain/eid/14014</td>
<td>/eid</td>
<td>&lt;source&gt;</td>
<td>No</td>
</tr>
<tr>
<td>Source List</td>
<td>/blp/mktlist/chain/source/UN;secclass=Equity</td>
<td>/source</td>
<td>&lt;source&gt;</td>
<td>No</td>
</tr>
</tbody>
</table>

a. The FLDS <GO> identifier associated with the expected key values for that particular topic is listed, where applicable, which can be found on FLDS <GO> on the Bloomberg Professional service.
b. Denotes whether that particular subscription (based on the <topic type> of the subscription string) will refresh and at what periodicity. For Daily refreshes, this will occur at the start of a new market day.

Here is a quick reference for the above FLDS <GO> identifiers:

<table>
<thead>
<tr>
<th>FLDS &lt;GO&gt; Identifier</th>
<th>Mnemonic</th>
<th>FLDS &lt;GO&gt; Identifier</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>DX194</td>
<td>PARSEKYABLE DES SOURCE</td>
<td>ID005</td>
<td>ID ISIN</td>
</tr>
<tr>
<td>DX282</td>
<td>FEED SOURCE</td>
<td>ID032</td>
<td>ID CUSIP</td>
</tr>
<tr>
<td>DY003</td>
<td>ID_BB_SEC_NUM DES</td>
<td>ID059</td>
<td>ID BB UNIQUE</td>
</tr>
<tr>
<td>EX005</td>
<td>ID EXCH_SYMBOL</td>
<td>ID122</td>
<td>ID BB SEC NUM SRC</td>
</tr>
<tr>
<td>ID002</td>
<td>ID SEDOL1</td>
<td>ID035</td>
<td>ID BB GLOBAL</td>
</tr>
</tbody>
</table>
The following code snippet demonstrates how to subscribe for streaming market list chain data and assumes that a session already exists and that the "/blp/mktlist" service has been successfully opened.

```c
const char *security = "/blp/mktlist.chain/esym/LN/BP";
SubscriptionList subscriptions;
subscriptions.add(security, CorrelationId((char *)security));
session.subscribe (subscriptions);
```
Response Overview

The Market List response will be a series of SUBSCRIPTION_DATA events, which you will be familiar with if you have developed Bloomberg API applications using any of the other streaming services, such as //blp/mktdata, //blp/mktvwap or //blp/mktdepth.

A SUBSCRIPTION_DATA event message will either be of type ListRecap or ListData. The initial such event message(s) will be of type ListRecap. These represent the initial paint of your chain of instruments. Within a single ListRecap message, you will find a LIST_LISTTYPE, comprising zero, or more, LIST_INSERT_ENTRIES.

If a subscription is made for a chain that does not contain any members, an empty list will be returned. An example of this is requesting the options for an equity that does not have any options. Although, there are no options for the equity, the subscription succeeds and a single ListRecap message will be received with LIST_INSERT_ENTRIES[] showing no elements. If the LIST_MUTABLE field value, from the ListRecap message is equal to 'MUTABLE', then that means there could be ListData items received later on, so you may wish to keep the subscription alive. The newly created members are then added to the previously empty list. However, if the LIST_MUTABLE field is 'IMMUTABLE', then that means it will not return any further updates and you may wish to terminate the subscription by unsubscribing. This is explained further in the following paragraph.

Various types of lists are available for a subscription. Though the subscription formats are the same, the lists could be:

- **ORDERED**: When a list is subscribed and the LIST_ORDERED field within the ListRecap message equals 'ORDERED', the items on the list are returned in ordered format.

- **UNORDERED**: When a list is subscribed and the LIST_ORDERED field within the ListRecap message equals 'NOTORDERED', the returned list of instruments could be in any order.

Similarly, a list subscription can be:

- **MUTABLE**: If the LIST_MUTABLE field within the ListRecap message equals 'MUTABLE', the constituent instruments of a list can change. All subsequent updates will be received as ListData messages.

- **IMMUTABLE**: If the LIST_MUTABLE field within the ListRecap message equals 'IMMUTABLE', the list of instruments will never change.
Table 9-6: List Actions

<table>
<thead>
<tr>
<th>ListAction Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>Delete all of the existing list members. This implies there is more data to come.</td>
</tr>
<tr>
<td>ADD</td>
<td>Add all of the list members in this set</td>
</tr>
<tr>
<td>CLEAR_AND_ADD</td>
<td>Delete all of the existing list members and then Add all of the list members in this sequence</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete all of the list members in this set. Member Identifiers must match the current Member Identifiers exactly.</td>
</tr>
<tr>
<td>END</td>
<td>This is the last set in the sequence.</td>
</tr>
<tr>
<td>CLEAR_AND_END</td>
<td>Delete all of the existing list members, as there are no more entries to follow (i.e. the list is empty)</td>
</tr>
<tr>
<td>ADD_AND_END</td>
<td>Add all of the list members in this set and end. There are no more entries in this sequence</td>
</tr>
<tr>
<td>CLEAR_AND_ADD_AND_END</td>
<td>Delete all of the existing list members, add this entry and end. There are no more entries in this sequence</td>
</tr>
<tr>
<td>DELETE_AND_END</td>
<td>Delete all of the list members in this set. Identifiers must match the current Member Identifiers exactly. Then end, as there are no more entries in this sequence.</td>
</tr>
</tbody>
</table>

Data Response For a "chain" Subscription

Here is sample Market List Chain output (A few entries from the beginning and end of a ListRecap message, along with one ListData message) for a Market List subscription to "// blp/mktlist/chain/source/TQ":

---

9 B-Pipe 145
In the above sample output, a *ListRecap* message was returned first with a large number of list entries (only the partial recap is shown, however) and a single *ListData* message, which is
an actual update to the subscription. Although, the ListRecap does not possess a LIST_ACTION value, you are to treat such a message as a CLEAR_AND_ADD action. In other words, you will clear your cache and add the entries included in the message.

In the ListRecap message, you will notice a few other pieces of information in addition to the entries, such as the LIST_LISTTYPE field (in our case, its value is "Source List", which you will find included in the "TABLE OF SUBSERVICE NAME EXAMPLES" shown earlier in this section), the EID and the LIST_MUTABLE value, which is MUTABLE in this case. This indicates that the constituent instruments of a list can change.

After the ListRecap message, you will see one such change to the list, which is returned in the form of a ListData message. This message includes the LIST_ACTION, among other fields. In this case, it is indicating that you will ADD this message to your list at the END (as indicated by ADD_AND_END).

Handling Multiple Messages (a.k.a. Fragments)

The summary (initial paint) messages can be split into one or more smaller messages in the case where the returned data is too large to fit into a single message. It will be up to you to handle this in your application.

You will achieve this by checking the Fragment type of any SUBSCRIPTION_DATA event ListRecap message. The Fragment enum is used to indicate whether a message is a fragmented message or not and what position it occurs within the chain of split fragmented messages. If the ListRecap is split into two parts, then the first message will have a Fragment type value of FRAGMENT_START and a last message of FRAGMENT_END. If the ListRecap is split into more than 2 parts, all middle Fragments will be of type FRAGMENT_INTERMEDIATE.

To check for the Fragment Type, you will call the fragmentType property of the Message object (e.g. msg.fragmentType()).

Within your application, you will check to see if the fragment type of the ListRecap message is FRAGMENT_NONE or FRAGMENT_START. If one of these are determined, then you will want to clear your list and begin adding the entries included in that part of the ListRecap message. In the case where FRAGMENT_START is determined, then you will know to continue reading the ListRecap messages and adding the entries to your list from those messages until you receive a ListRecap with a fragment type for FRAGMENT_END. At this point, you know you are finished building your list and it is now time to wait for any subsequent ListData updates.
Snapshot Request For List Of Security Identifiers

If you would like to retrieve a list of all available sources that are pricing a given instrument, then you will use the 'secids' subservice. This request is particularly useful when the original subscription string provided by the client triggers a 'NOTUNIQUE' response from the service. Using this subservice, you also have the ability to filter your results to only a particular source.

The following table lists all of the Topic Types, their applicable topic key formats and associated B-Pipe mnemonic and FLDS <GO> field identifiers.

Table 9-8: Supported <Topic Type> Values

<table>
<thead>
<tr>
<th>Topic Type</th>
<th>Topic Key</th>
<th>B-PIPE Field</th>
<th>FLDS&lt;GO&gt; Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bpkbl</td>
<td>/&lt;identifier&gt;</td>
<td>PARSEKYABLE_DES_SOURCE</td>
<td>DX194 and DS587</td>
</tr>
<tr>
<td>/bsid</td>
<td>/&lt;identifier&gt;</td>
<td>ID_BB_SEC_NUM_SRC</td>
<td>ID122</td>
</tr>
<tr>
<td>/bsym</td>
<td>/&lt;identifier&gt;</td>
<td>ID_BB_SEC_NUM_DES</td>
<td>DY003</td>
</tr>
<tr>
<td>/buid</td>
<td>/&lt;identifier&gt;</td>
<td>ID_BB_UNIQUE</td>
<td>ID059</td>
</tr>
<tr>
<td>/cusip</td>
<td>/&lt;identifier&gt;</td>
<td>ID_CUSIP</td>
<td>ID032</td>
</tr>
<tr>
<td>/esym</td>
<td>/&lt;identifier&gt;</td>
<td>ID_EXCH_SYMBOL</td>
<td>EX005-&gt;EX011</td>
</tr>
<tr>
<td>/isin</td>
<td>/&lt;identifier&gt;</td>
<td>ID_ISIN</td>
<td>ID005</td>
</tr>
<tr>
<td>/sedol</td>
<td>/&lt;identifier&gt;</td>
<td>ID_SEDOL1</td>
<td>ID002</td>
</tr>
<tr>
<td>/bbgid</td>
<td>/&lt;identifier&gt;</td>
<td>ID_BB_GLOBAL</td>
<td>ID135</td>
</tr>
<tr>
<td>/ticker</td>
<td>/&lt;identifier&gt;</td>
<td>PARSEKYABLE_DES_SOURCE</td>
<td>DX194 and DS587</td>
</tr>
</tbody>
</table>

Market list requests with the secids subservice name are always IMMUTABLE, which means that the returned list of instruments does not receive update messages and must be re-requested to discover any new pricing sources that emerge after the initial request.

Table 9-9: Market List Requests with the Secids Subservice Name

<table>
<thead>
<tr>
<th>Key Field</th>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bloomberg Unique ID</td>
<td>/blp/mktlist/secids/buid/uniqueid</td>
<td>All instrument IDs for the given buid</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/buid/EQ0010080100001000</td>
<td></td>
</tr>
<tr>
<td>Bloomberg Symbol</td>
<td>/blp/mktlist/secids/bsym/symbol</td>
<td>All instrument IDs for the given bsym</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/bsym/VOD</td>
<td></td>
</tr>
<tr>
<td>SEDOL</td>
<td>/blp/mktlist/secids/sedol/sedol</td>
<td>All instrument IDs for the given SEDOL</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/sedol/2005973</td>
<td></td>
</tr>
<tr>
<td>CUSIP</td>
<td>/blp/mktlist/secids/cusip/cusip</td>
<td>All instrument IDs for the given CUSIP</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/cusip/459200101</td>
<td></td>
</tr>
<tr>
<td>ISIN</td>
<td>/blp/mktlist/secids/isin/isin</td>
<td>All instrument IDs for the given ISIN</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/isin/US4592001014</td>
<td></td>
</tr>
<tr>
<td>Parsekeyable</td>
<td>/blp/mktlist/secids/bpkbl/parsekeyable</td>
<td>All instrument IDs for the given parsekeyable</td>
</tr>
<tr>
<td></td>
<td>/blp/mktlist/secids/bpkbl/UKX Index</td>
<td></td>
</tr>
</tbody>
</table>
A security-based secids request can also be modified to limit the source using the 'source' parameter. This table demonstrates such an instrument with and without the "source" parameter.

Table 9-10: SecidsRequests with and without Source Parameter

<table>
<thead>
<tr>
<th>Subscription String</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>//blp/mktlist/secids/cusip/459200101</td>
<td>This example returns all IDs for the given CUSIP.</td>
</tr>
<tr>
<td>//blp/mktlist/secids/cusip/459200101;source=US</td>
<td>This example returns all IDs for the given CUSIP, but limited to source US.</td>
</tr>
</tbody>
</table>

The following code snippet demonstrates how to request static market list snapshot data and assumes that a session already exists and that the "//blp/mktlist" service has been successfully opened.

```c
const char *security = "//blp/mktlist/secids/cusip/459200101;source=US";
Service mktListService = session.getService("//blp/mktlist");
Request request = mktListService.createRequest("SnapshotRequest");
request.set("security", security);
```
Data Response For "secids" Snapshot Request

The following data response is associated with the snapshot request code snippet.

```
SnapshotRequest = { security = //blp/mktlist/secids/cusip/459200101;source=US }
LIST_ID = //blp/mktlist/secids/cusip/459200101;source=US
EID = 35009
LIST_LISTTYPE = Security IDs
LIST_INSERT_ENTRIES
  ID_BB_SEC_NUM_SRC = 399432473346
  FEED_SOURCE = US
  ID_BB_SEC_NUM_DES = IBM
  ID_BB_UNIQUE = EQ0010080100001000
  SECURITY_TYP2 = Equity
LIST_ORDERED = NOTORDERED
LIST_MUTABLE = IMMUTABLE
```

In your application, you will handle the data response the same way, initially, as you would any static request. This is accomplished by checking the event type of the incoming message. If its event type is PARTIAL_RESPONSE, then that indicates that there is at least one more message to be received to fulfill that request. You will continue reading the incoming messages until you receive a RESPONSE event type, which indicates that the request has been fully served. Further information is available in "Appendix A Schemas".
Here is a sample event handler written in C++. It was extracted from the "MarketListSnapshotExample" example found in the B-Pipe C++ API SDK, and is the event handler that is responsible for displaying the above output to a console window.

```cpp
void eventLoop(Session &session)
{
    bool done = false;
    while (!done) {
        Event event = session.nextEvent();
        if (event.eventType() == Event::PARTIAL_RESPONSE) {
            std::cout << "Processing Partial Response" << std::endl;
            processResponseEvent(event);
        }
        else if (event.eventType() == Event::RESPONSE) {
            std::cout << "Processing Response" << std::endl;
            processResponseEvent(event);
            done = true;
        } else {
            MessageIterator msgIter(event);
            while (msgIter.next()) {
                Message msg = msgIter.message();
                if (event.eventType() == Event::SESSION_STATUS) {
                    if (msg.messageType() == SESSION_TERMINATED ||
                        msg.messageType() == SESSION_STARTUP_FAILURE) {
                        done = true;
                    }
                }
            }
        }
    }
}
```
If you examine the response from the example market list request, which is "/blp/mktlist/secids/cusip/459200101;source=US", you will find that the data is all returned in a single message, which means that the message will have an event type of "RESPONSE". Within that block of code, there is a call to processResponseEvent(). It is here that we first check for the responseCode element. To understand why we are checking for this element, you will first need to understand the structure of the schema for the //blp/mktlist service. Here is a screenshot capturing the sub-elements of the SnapshotRequest/Responses node.
If the responseCode is found in the message, then you will check to see if the resultCode is greater than zero. If it is, then this is an indication that there was a problem with the request and that this message contains an error. The details of the error will be provided by the message's responseCode, resultText and sourceId values.

If the resultCode equals zero, then data can be expected to be contained within the message. In this case, we will retrieve the snapshot element of the message. You will see in the above processResponseEvent() handler that the number of elements contained in the snapshot are determined by a call to numElements() and then each of those elements are then read into a dataItem variable, of type Element, one at a time. You can check to see if the dataItem is an array by calling its isArray() function. If it returns true, then it is an array containing one, or more, items and must be processed differently than if containing a single item.

You will see in the schema screenshot that there are a total of ten possible single field elements and one array element in a snapshot. The array element is indicated by the SEQUENCE type. In our case, the resultCode is zero (i.e. no errors) and there are 6 elements contained in the snapshot element. The first 3 of them are single field elements (e.g. LIST_ID, EID, LIST_LISTTYPE), which means that isArray() returns false for each of them. However, the 4th element, LIST_INSERT_ENTRIES, is an array (a.k.a. SEQUENCE type). This element is then processed in the processBulkData() function. The remaining two elements (LIST_ORDERED and LIST_MUTABLE) are also single field elements.
9.2.3 Source Reference Service

Overview

The Source Reference and Tick Size subscription service (aka //blp/srcref) is used to subscribe to the source reference and tick size data available for the specified entitlement ID. Currently, this is available per EID (FEED_EID). This allows an application to retrieve the source reference/tick size information for all the EIDs it is entitled for. This service is available to both BPS (BLOOMBERG PROFESSIONAL Service) and Non-BPS users. The available source reference information includes:

- All possible values of FEED_SOURCE for the EID and a short description of the source
- Whether or not the source is a composite and all the local sources for composites
- All of the Broker codes and names
- All condition codes with a short description

The syntax of the Source Reference subscription string is as follows:

//<service owner>/<service name>/<subservice name>/<topic>

where <topic> is comprised of <topic type>/<topic key>. Table 9-11 provides further details.

Table 9-11: Source Reference String Definitions

<table>
<thead>
<tr>
<th>&lt;service owner&gt;</th>
<th>For B-Pipe is &quot;blp&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;service name&gt;</td>
<td>Source Reference and Tick Size subscription service name is &quot;/srcref&quot;</td>
</tr>
<tr>
<td>&lt;subservice name&gt;</td>
<td>/brokercodes, /conditioncodes, /tradingstatuses or /ticksizes (see Table 9-11.)</td>
</tr>
<tr>
<td>&lt;topic type&gt;</td>
<td>/eid</td>
</tr>
<tr>
<td>&lt;topic key&gt;</td>
<td>EID-Number (FEED_EID1 =&gt; FEED_EID4)</td>
</tr>
</tbody>
</table>

There are currently four subservices that can be used in your subscription string.

Table 9-12: Subservice Definitions

<table>
<thead>
<tr>
<th>Subservice</th>
<th>Subscription String Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/brokercodes</td>
<td>//blp/srcref/brokercodes/eid/&lt;eid&gt;</td>
<td>List of all possible broker codes for a specified EID</td>
</tr>
<tr>
<td>/conditioncodes</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;</td>
<td>List of Market Depth, Quote, and Trade condition codes for a specified EID</td>
</tr>
<tr>
<td>/tradingstatuses</td>
<td>//blp/srcref/tradingstatuses/eid/&lt;eid&gt;</td>
<td>List of trading statuses and trading periods for a specified EID.</td>
</tr>
<tr>
<td>/ticksizes</td>
<td>//blp/srcref/ticksizes/eid/&lt;eid&gt;</td>
<td>List of Tick Sizes for a specified EID.</td>
</tr>
</tbody>
</table>
Filters can be used for /conditioncodes and /tradingstatuses subscription only. Here are the possible filters available for each:

<table>
<thead>
<tr>
<th>Filter Name (type)</th>
<th>Subscription String Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subservice Name:</strong> /conditioncodes</td>
<td></td>
</tr>
<tr>
<td>TRADE</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=TRADE</td>
</tr>
<tr>
<td>QUOTE</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=QUOTE</td>
</tr>
<tr>
<td>MKTDEPTH</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=MKTDEPTH</td>
</tr>
<tr>
<td>TRADE,QUOTE</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=TRADE,QUOTE</td>
</tr>
<tr>
<td>TRADE,MKTDEPTH</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=TRADE,MKTDEPTH</td>
</tr>
<tr>
<td>QUOTE,MKTDEPTH</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=QUOTE,MKTDEPTH</td>
</tr>
<tr>
<td>TRADE,QUOTE,MKTDEPTH</td>
<td>//blp/srcref/conditioncodes/eid/&lt;eid&gt;?type=TRADE,QUOTE,MKTDEPTH</td>
</tr>
</tbody>
</table>

| **Subservice Name:** /tradingstatuses | |
| PERIOD              | //blp/srcref/tradingstatuses/eid/<eid>?type=PERIOD |
| STATUS              | //blp/srcref/tradingstatuses/eid/<eid>?type=STATUS |
| PERIOD,STATUS       | //blp/srcref/tradingstatuses/eid/<eid>?type=PERIOD,STATUS |

For subscriptions without a filter, users will receive all event types of that subservice name in the initial snapshot, as well as within subsequent daily updates. However, for subscriptions with filters, users will receive all events in the initial snapshot, but only specified events within subsequent daily updates.

**Important BPOD Upgrade Notes:**

1. B-Pipe breaks down the subscriptions into a more granular format. With BPOD, you would have subscribed to "/blp/mktref/srcref/eid/<eid>" to obtain all source references for that EID, which included the broker codes, trade condition codes, quote condition codes, market depth condition codes, period suspense codes, security suspense codes and ticksizes. Now, by using B-Pipe, you can break down these source references into four main subscriptions: "/blp/srcref/brokercodes/eid/<eid>", "/blp/srcref/conditioncodes/eid/<eid>", "/blp/srcref/tradingstatuses/eid/<eid>" and "/blp/srcref/ticksizes/eid/<eid>".

2. B-Pipe has introduced filters for some of its subservices to allow you to subscribe to the data you are most interested.

3. With B-Pipe, we return a description message for each subservice's sources.

4. With B-Pipe, Bloomberg now offers intraday updating for tick size changes.

5. If you are looking for the sources on contributor EIDs (or any EID), you should subscribe to /blp/srcref for any of the subservices (i.e. /ticksizes, /brokercode, etc) and the list of descriptions for that source will be included even if the subservice doesn't apply. For example, "/blp/srcref/ticksizes/eid/14240" will return the sources for 14240, but there will not be any ticksizes information.
Code Example

You will find a SourceRefSubscriptionExample in the B-Pipe SDK for C++, Java and .NET. This C++ example demonstrates how to make a simple Source Reference subscription for the condition codes associated with EID 14003.

```cpp
const char *list = "/blp/srcref/conditioncodes/eid/14003";
SubscriptionList subscriptions;
subscriptions.add(list, CorrelationId((char *)security));
session.subscribe (subscriptions);
```

Figure 9-4: C++ code snippet - subscribing for a list of condition codes for EID 14003

Response Overview

The Source Reference response will be a series of SUBSCRIPTION_DATA events, which you will be familiar with if you have developed Bloomberg API applications using any of the other streaming services, such as //blp/mktdata, //blp/mktlist or //blp/mktdepth.

All SUBSCRIPTION_DATA event messages will be of message type SourceReferenceUpdates and will contain a SOURCE_REF_EVENT_TYPE_RT (event type), SOURCE_REF_EVENT_SUBTYPE_RT (event sub-type) and EID field (int32), along with an array of event type field items applicable to the subservice you are subscribing. Table 9-14 lists the possible enumeration values for the event type and event sub-type fields:

Table 9-14: Enumeration Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE_REF_EVENT_TYPE_RT</td>
<td>This specifies the event type.</td>
<td>Possible enumeration values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BROKER_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRADE_CONDITION_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUOTE_CONDITION_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MKTDEPTH_CONDITION_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRADING_PERIOD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRADING_STATUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TICK_SIZE_TABLE</td>
</tr>
<tr>
<td>SOURCE_REF_EVENT_SUBTYPE_RT</td>
<td>This specifies the event sub-type</td>
<td>Possible enumeration values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INITPAINT - Initial Paint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REFRESH - Daily Refresha</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UPDATE - Intraday Update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a. Refreshes are performed daily at approximately 6pm (Eastern Standard Time).</td>
</tr>
</tbody>
</table>

The subservice name included in your subscription will dictate which event type (SOURCE_REF_EVENT_TYPE_RT) field items will be returned as initial snapshot.
Response Event Types by Subservice

Table 9-15 lists all of the initial snapshot and refresh (i.e., INITPAINT and REFRESH, respectively) event type fields you should expect to receive for the subservice you are subscribing.

<table>
<thead>
<tr>
<th>Subservice Name</th>
<th>Response Event Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>/brokercodes</td>
<td>DESCRIPTION + BROKER_CODE</td>
</tr>
<tr>
<td>/conditioncodes</td>
<td>DESCRIPTION + TRADE_COND_CODE + QUOTE_COND_CODE + MKTDEPTH_COND_CODE</td>
</tr>
<tr>
<td>/tradingstatuses</td>
<td>DESCRIPTION + TRADING_PERIOD + TRADING_STATUS</td>
</tr>
<tr>
<td>/ticksizes(^a)</td>
<td>DESCRIPTION + TICK_SIZE_TABLE</td>
</tr>
</tbody>
</table>

\(^a\) All subservices will return INITPAINT and REFRESH event messages. However, /ticksizes will also return UPDATE event messages.

For a breakdown of each message returned for your subservice, please see Table 9-16.

Breakdown of Event Type Fields

Table 9-16 describes the breakdown of each event type's field array. Each name given to the field array is the pluralized form of the aforementioned event type value (e.g., The DESCRIPTION event type value (as found in Table 9-15) will have an associated field array name of DESCRIPTIONS).

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTIONS</td>
<td>SourceReferenceDescriptions</td>
<td>Contains the feed EID and feed source, along with a list of DESCRIPTION entries containing each item's expanded name of the data contributor or exchange and local source of the composite source for lookup to condition code and broker.</td>
</tr>
<tr>
<td>BROKER_CODES</td>
<td>SourceReferenceBrokerCodes</td>
<td>Contains the feed EID and feed source, along with a list of BROKER_CODE entries containing each item's Bloomberg mnemonic and associated name.</td>
</tr>
<tr>
<td>TRADE_COND_CODES</td>
<td>SourceReferenceTradeConditionCodes</td>
<td>Contains the feed EID and feed source, along with a list of TRADE_COND_CODE entries containing each item's Bloomberg mnemonic(s) for special conditions on the trade, condition code, trade category, short name for the sale condition, ESMA transaction code and more.</td>
</tr>
</tbody>
</table>
Handling Multiple Messages (a.k.a. Fragments)

- As you will see, initial paint messages can be split into one or more smaller messages in the case where the returned data is too large to fit into a single message. It will be up to you to handle this in your application.

- You will achieve this by checking the Fragment type of any SUBSCRIPTION_DATA event SourceReferenceUpdates message. The Fragment enum is used to indicate whether a message is a fragmented message or not and what position it occurs within the chain of split fragmented messages. If the SourceReferenceUpdates is

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUOTE_COND_CODES</td>
<td>SourceReferenceQuoteConditionCodes</td>
<td>Contains the feed EID and feed source, along with a list of QUOTE_COND_CODE entries containing each item's quote condition mnemonic, Bloomberg condition code, quote condition short name and Provider assigned condition code mnemonic(s).</td>
</tr>
<tr>
<td>MKTDEPTH_COND_CODES</td>
<td>SourceReferenceMarketDepthConditionCodes</td>
<td>Contains the feed EID and feed source, along with a list of MKTDEPTH_COND_CODE entries containing each item's Bloomberg mnemonic, for the condition, short name for the condition and Provider assigned condition code mnemonic(s).</td>
</tr>
<tr>
<td>TRADING_PERIODS</td>
<td>SourceReferenceTradingPeriods</td>
<td>Contains the feed EID and feed source, along with a list of TRADING_PERIOD entries containing each item's Bloomberg assigned mnemonic for the current trading period of a security, Bloomberg's short name for the current trading period of the security, and Bloomberg's assigned simplified status mnemonic for the current market status of a security.</td>
</tr>
<tr>
<td>TRADING_STATUSES</td>
<td>SourceReferenceTradingStatuses</td>
<td>Contains the feed EID and feed source, along with a list of TRADING_PERIOD entries containing each item's Bloomberg assigned mnemonic for the current trading status of a security, Bloomberg's short name for the market status on a source, and Bloomberg's assigned simplified status mnemonic for the current market status of a security.</td>
</tr>
<tr>
<td>TICK_SIZE_TABLES</td>
<td>TickSizeTable</td>
<td>Contains the feed EID, feed source, table field name, table identifier, percent field name, table type and frequency at which the tick size can change, along with a list of TICK_SIZE_TABLE_ROW entries containing each item's type of tick size value, lower/upper bounds value, and tick size value used for the range.</td>
</tr>
</tbody>
</table>
split into two parts, then the first message will have a Fragment type value of 
FRAGMENT_START and a last message of FRAGMENT_END. If the 
SourceReferenceUpdates is split into more than 2 parts, all middle Fragments will 
be of type FRAGMENT_INTERMEDIATE.

Table 9-17: Fragment Type Enumerators

<table>
<thead>
<tr>
<th>Fragment Type Enumerators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGMENT_NONE</td>
<td>Message is not fragmented</td>
</tr>
<tr>
<td>FRAGMENT_START</td>
<td>The first fragmented message</td>
</tr>
<tr>
<td>FRAGMENT_INTERMEDIATE</td>
<td>Intermediate fragmented messages</td>
</tr>
<tr>
<td>FRAGMENT_END</td>
<td>The last fragmented message</td>
</tr>
</tbody>
</table>

Data Response For Subscription

Here is sample output for a Source Reference subscription to "/blp/srcref/ticksizes/eid/14014":

```
* INITIAL SNAPSHOT

SourceReferenceUpdates = {
    SOURCE_REF_EVENT_TYPE_RT = DESCRIPTION
    SOURCE_REF_EVENT_SUBTYPE_RT = INITPAINT
    EID = 35009
    DESCRIPTIONS[] =
        DESCRIPTIONS = {
            FEED_SOURCE = LN
            FEED_EID = 14014
            DESCRIPTION[] =
                DESCRIPTION = {
                    FEED_SOURCE_DES_RT = London Stock Exchange Domestic
                }
        }
    -- MORE --
}

SourceReferenceUpdates = {
    SOURCE_REF_EVENT_TYPE_RT = TICK_SIZE_TABLE
    SOURCE_REF_EVENT_SUBTYPE_RT = INITPAINT
    EID = 35009
    TICK_SIZE_TABLES[] =
        TICK_SIZE_TABLES = {
            FEED_SOURCE = LN
            FEED_EID = 14014
            TICK_SIZE_TABLE_IDENTIFIER_RT = 2871
            TICK_SIZE_TABLE_TYPE_RT = PRICE
            TICK_SIZE_TABLE_UPDATE_FREQ_RT = DAILY
            TICK_SIZE_TABLE_FIELD_NAME_RT = LAST_TRADE
        }
```
TICK_SIZE_TABLE_ROW[] = 
TICK_SIZE_TABLE_ROW = {
  TICK_SIZE_TABLE_PRICE_TYPE_RT = ABSOLUTE
  TICK_SIZE_TBL_BAND_TICK_SIZE_RT = 0.000100
  TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 0.000100
  TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 10.000000
}
TICK_SIZE_TABLE_ROW = {
  TICK_SIZE_TABLE_PRICE_TYPE_RT = ABSOLUTE
  TICK_SIZE_TBL_BAND_TICK_SIZE_RT = 0.010000
  TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 10.000000
  TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 100.000000
}

-- MORE --

***********************************************************************
* DAILY REFRESH
***********************************************************************
SourceReferenceUpdates = {
  SOURCE_REF_EVENT_TYPE_RT = DESCRIPTION
  SOURCE_REF_EVENT_SUBTYPE_RT = REFRESH
  EID = 35009
  DESCRIPTIONS[] = 
    DESCRIPTIONS = {
      FEED_SOURCE = LN
      FEED_EID = 14014
      DESCRIPTION[] = 
        DESCRIPTION = {
          FEED_SOURCE_DES_RT = London Stock Exchange Domestic
        }
    }
-- MORE --
}
SourceReferenceUpdates = {
  SOURCE_REF_EVENT_TYPE_RT = TICK_SIZE_TABLE
  SOURCE_REF_EVENT_SUBTYPE_RT = REFRESH
  EID = 35009
  TICK_SIZE_TABLES[] = 
    TICK_SIZE_TABLES = {
      FEED_SOURCE = LN
      FEED_EID = 14014
      TICK_SIZE_TABLE_IDENTIFIER_RT = 5977
      TICK_SIZE_TABLE_TYPE_RT = PRICE
      TICK_SIZE_TABLE_UPDATE_FREQ_RT = DAILY
      TICK_SIZE_TABLE_FIELD_NAME_RT = LAST_TRADE
      TICK_SIZE_TABLE_ROW[] = 
        TICK_SIZE_TABLE_ROW = {
          TICK_SIZE_TABLE_PRICE_TYPE_RT = ABSOLUTE
          TICK_SIZE_TBL_BAND_TICK_SIZE_RT = 0.000100
          TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 0.000100
          TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 10.000000
        }
        TICK_SIZE_TABLE_ROW = {
          TICK_SIZE_TABLE_PRICE_TYPE_RT = ABSOLUTE
          TICK_SIZE_TBL_BAND_TICK_SIZE_RT = 0.010000
          TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 10.000000
          TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 100.000000
        }
    }
-- MORE --
}
In the above sample output, a subscription containing the subservice "/ticksizes" was made, which means that you can expect to receive "INITPAINT" and "REFRESH" event types (i.e. SOURCE_REF_EVENT_TYPE_RT) messages containing "DESCRIPTION" and "TICK_SIZE_TABLE" event sub-types (i.e. SOURCE_REF_EVENT_SUBTYPE_RT). In addition to the aforementioned messages, which are standard for all of the subservice requests, you will also receive "UPDATE" event type messages, which are unique to the /ticksizes subservice. However, there will not be an UPDATE "DESCRIPTION" message sent.

Taking a look at the sample output above, you will notice that every SourceReferenceUpdates message contains the standard event type, sub-type and EID single-value fields, along with an array of fields applicable for that event type. For instance, in the message containing the event type of "TICK_SIZE_TABLE" you will find an array of "TICK_SIZE_TABLES" fields.
A Schemas

A.1 Overview

Each of the following sections provides an overview of the request options and response structure for each request type within each of the Bloomberg API services. A service is defined by a request and a response schema. In the following sections the request schema is broken into tables detailing all options and arguments and example syntax. The response schema is represented graphically.

A.2 Reference Data Service //blp/refdata

Note: B-PIPE supports only the ReferenceDataRequest type on the Reference Data Service. All other request types on the ReferenceDataService are not supported by B-PIPE.

A.2.1 Operations

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Request Type</th>
<th>Response Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HistoricalDataRequest</td>
<td>HistoricalDataRequest</td>
<td>HistoricalDataResponse</td>
<td>Request Historical Data</td>
</tr>
<tr>
<td>IntraDayTickRequest</td>
<td>IntraDayTickRequest</td>
<td>IntraDayTickResponse</td>
<td>Request Intraday Tick Data</td>
</tr>
<tr>
<td>IntraDayBarRequest</td>
<td>IntraDayBarRequest</td>
<td>IntradayBarResponse</td>
<td>Request Intraday Bar Data</td>
</tr>
<tr>
<td>ReferenceDataRequest</td>
<td>ReferenceDataRequest</td>
<td>ReferenceDataResponse</td>
<td>Request Reference Data</td>
</tr>
<tr>
<td>PortfolioDataRequest</td>
<td>PortfolioDataRequest</td>
<td>PortfolioDataResponse</td>
<td>Request Portfolio Data</td>
</tr>
<tr>
<td>BeqsRequest</td>
<td>BeqsRequest</td>
<td>BeqsResponse</td>
<td>Request EQS Screen Data</td>
</tr>
</tbody>
</table>

A.2.2 ReferenceDataRequest: Sequence

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>securities</td>
<td>string array</td>
<td>string</td>
<td>See &quot;Security/Securities&quot; on page 77 for additional details.</td>
</tr>
</tbody>
</table>

Example Syntax: Element securities = request.GetElement("securities"); securities.AppendValue("VOD LN Equity");

Fields: the reference fields desired which correspond to data points. See FLDS<GO> for a list of more information.
<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>string</td>
<td></td>
<td>See “Fields” on page 79 for additional details.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
Element fields = request.GetElement("fields");
fields.AppendValue("PX_LAST");
```

**Overrides:** Append overrides to modify the calculation

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldID</td>
<td>string</td>
<td>field mnemonic, PRICING_SOURCE, or field alpha-numeric, PR092. Review FLDS&lt;GO&gt; for list of possible overrides.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>the desired override value</td>
<td></td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
Element overrides = request["overrides"];
Element override1 = overrides.AppendElement();
override1.SetElement("fieldId", "PRICING_SOURCE");
override1.SetElement("value", "CG");
```

**Return Entitlements:** returns the entitlement identifiers associated with security.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnEids</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting this to true will populate fieldData with an extra element containing a name and value for the EID date.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
request.Set("returnEids", true);
```

**Return Formatted Value:** returns all data as a data type string

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnFormattedValue</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting to true will force all data to be returned as a string.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
request.Set("returnFormattedValue", true);
```

**Use UTC Time:** return date and time values as Coordinated Universal Time (UTC) values

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useUTCTime</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting to true returns values in UTC. Setting this to false will default to the TZDF&lt;GO&gt; settings of the requestor.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
request.Set("useUTCTime", true);
```

**Forced Delay:** returns the latest reference data up to the delay period.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forcedDelay</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting to true will return the latest data up to the delay period specified by the exchange for this security. For example requesting VOD LN Equity and PX_LAST will return a snapshot of the last price from 15mins ago.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```csharp
request.Set("forcedDelay", true);
```
A.2.3 ReferenceDataResponse: Choice

Figure A-1 provides the structure of a ReferenceDataResponse. See “Reference Data Service Response” on page 182 for more information.

Figure A-1: Reference Data Request Response
### A.2.4 HistoricalDataRequest: Sequence

<table>
<thead>
<tr>
<th>Securities: A stock or bond.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>securities</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```csharp
Element securities = request.GetElement("securities");
securities.AppendValue("VOD LN Equity");
```

<table>
<thead>
<tr>
<th>Fields: the reference fields desired which correspond to data points. See FLDS&lt;GO&gt; for a list of more information.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>fields</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```csharp
Element fields = request.GetElement("fields");
fields.AppendValue("PX_LAST");
```

<table>
<thead>
<tr>
<th>Start Date: the first date of the period to retrieve data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>startDate</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```csharp
request.Set("startDate", "20090601");
```

<table>
<thead>
<tr>
<th>End Date: the end date of the period to retrieve data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>endDate</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```csharp
request.Set("endDate", "20100601");
```

<table>
<thead>
<tr>
<th>Period Adjustment: Determine the frequency and calendar type of the output. To be used in conjunction with Period Selection.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>periodicityAdjustment</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Example Syntax:**
```csharp
request.Set("periodicityAdjustment", "ACTUAL");
```
**Period Selection:** Determine the frequency of the output. To be used in conjunction with Period Adjustment.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>periodicitySelection</td>
<td>DAILY</td>
<td>string</td>
<td>Returns one data point per day</td>
</tr>
<tr>
<td></td>
<td>WEEKLY</td>
<td>string</td>
<td>Returns one data point per week</td>
</tr>
<tr>
<td></td>
<td>MONTHLY</td>
<td>string</td>
<td>Returns one data point per month</td>
</tr>
<tr>
<td></td>
<td>QUARTERLY</td>
<td>string</td>
<td>Returns one data point per quarter</td>
</tr>
<tr>
<td></td>
<td>SEMI_ANNUALLY</td>
<td>string</td>
<td>Returns one data point per half year</td>
</tr>
<tr>
<td></td>
<td>YEARLY</td>
<td>string</td>
<td>Returns one data point per year</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("periodicitySelection", "DAILY");`

**Currency:** Amends the value from local to desired currency

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currency</td>
<td>Currency of the ISO code, e.g., USD, GBP</td>
<td>string</td>
<td>The 3 letter ISO code. View WCV&lt;GO&gt; on the BLOOMBERG PROFESSIONAL service for a list of currencies.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("currency", "USD");`

**Override Options:** Indicates whether to use the average or the closing price in quote calculation.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overrideOption</td>
<td>OVERRIDE_OPTION_CLOS E</td>
<td>string</td>
<td>Use the closing price in quote calculation</td>
</tr>
<tr>
<td></td>
<td>OVERRIDE_OPTION_GPA</td>
<td>string</td>
<td>Use the average price in quote calculation</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("overrideOption", "OVERRIDE_OPTION_GPA");`

**Pricing Options:** Sets quote to Price or Yield for a debt instrument whose default value is quoted in yield (depending on pricing source).

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pricingOption</td>
<td>PRICING_OPTION_PRICE</td>
<td>string</td>
<td>Set quote to price</td>
</tr>
<tr>
<td></td>
<td>PRICING_OPTION_YIELD</td>
<td>string</td>
<td>Set quote to yield</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("pricingOption", "PRICING_OPTION_PRICE");`

**Non Trading Day Fill Option:** Sets to include/exclude non trading days where no data was generated.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonTradingDayFillOption</td>
<td>NON_TRADING_WEEKDAYS</td>
<td>string</td>
<td>Include all weekdays (Monday to Friday) in the data set</td>
</tr>
<tr>
<td></td>
<td>ALLCALENDAR_D AYS</td>
<td>string</td>
<td>Include all days of the calendar in the data set returned</td>
</tr>
<tr>
<td></td>
<td>ACTIVE_DAYS_ONLY</td>
<td>string</td>
<td>Include only active days (days where the instrument and field pair updated) in the data set returned</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("nonTradingDayFillOption", "NON_TRADING_WEEKDAYS");`
**Non Trading Day Fill Method:** If data is to be displayed for non trading days what is the data to be returned.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonTradingDayFillMethod</td>
<td>PREVIOUS_VALUE</td>
<td>string</td>
<td>Search back and retrieve the previous value available for this security field pair. The search back period is up to one month.</td>
</tr>
<tr>
<td></td>
<td>NIL_VALUE</td>
<td>string</td>
<td>Returns blank for the &quot;value&quot; value within the data element for this field.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("nonTradingDayFillMethod", "PREVIOUS_VALUE");`

**Max Data Points:** the maximum number of data points to return.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxDataPoints</td>
<td></td>
<td>integer</td>
<td>The response will contain up to X data points, where X is the integer specified. If the original data set is larger than X, the response will be a subset, containing the last X data points. Hence the first range of data points will be removed.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("maxDataPoints", 100);`

**Return Entitlements:** returns the entitlement identifiers associated with security.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnEids</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting this to TRUE will populate fieldData with an extra element containing a name and value for the EID date.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("returnEIDs", true);`

**Return Relative Date:** returns data with a relative date.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnRelativeDate</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting this to true will populate fieldData with an extra element containing a name and value for the relative date. For example RELATIVE_DATE = 2002 Q2</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("returnRelativeDate", true);`

**Adjustment Normal:** Adjust for "change on day"

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentNormal</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing to reflect: Regular Cash, Interim, 1st Interim, 2nd Interim, 3rd Interim, 4th Interim, 5th Interim, Income, Estimated, Partnership Distribution, Final, Interest on Capital, Distribution, Prorated.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("adjustmentNormal", true);`
### Adjustment Abnormal: Adjusts for Anormal Cash Dividends

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentAbnormal</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing to reflect: Special Cash, Liquidation, Capital Gains, Long-Term Capital Gains, Short-Term Capital Gains, Memorial, Return of Capital, Rights Redemption, Miscellaneous, Return Premium, Preferred Rights Redemption, Proceeds/Rights, Proceeds/Shares, Proceeds/Warrants.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("adjustmentAbnormal", true);`

### Adjustment Split: Capital Changes Defaults

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentSplit</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing and/or volume to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, Rights Offerings/Entitlement.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("adjustmentSplit", true);`

### Adjustment Follow DPDF: Follow the BLOOMBERG PROFESSIONAL service function DPDF<GO>

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentFollowDPDF</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting to true will follow the DPDF&lt;GO&gt; BLOOMBERG PROFESSIONAL service function. True is the default setting for this option.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("adjustmentFollowDPDF", true);`

### CalendarCodeOverride: Returns the data based on the calendar of the specified country, exchange, or religion.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calendarCodeOverride</td>
<td>CDR &lt;GO&gt; calendar type</td>
<td>String</td>
<td>Returns the data based on the calendar of the specified country, exchange, or religion from CDR&lt;GO&gt;. Taking a two character calendar code null terminated string. This will cause the data to be aligned according to the calendar and including calendar holidays. Only applies only to DAILY requests.</td>
</tr>
</tbody>
</table>

**Example Syntax:** `request.Set("calendarCodeOverride", "US");`
**CalendarOverridesInfo**: Returns data based on the calendar code of multiple countries, exchanges, or religious calendars from CDR<GO>.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calendarOverrides</td>
<td>CDR&lt;GO&gt;</td>
<td>calendar type String array</td>
<td>Accepts a two-character calendar code null-terminated string of multiple country, exchange, or religious calendars from CDR&lt;GO&gt;. This will cause the data to be aligned according to the set calendar(s) including their calendar holidays. Only applies to DAILY requests.</td>
</tr>
<tr>
<td>calendareOverridesOperation</td>
<td>CDR_AND</td>
<td>String</td>
<td>Default value. Returns the intersection of trading days. That is, a data point is returned if a date is a valid trading day in all calendar codes specified in the request.</td>
</tr>
<tr>
<td></td>
<td>CDR_OR</td>
<td>String</td>
<td>Returns the union of trading days. That is, a data point is returned if a date is a valid trading day for any of the calendar codes specified in the request.</td>
</tr>
</tbody>
</table>

**Example Syntax**: Element cdrOverridesInfo = request.GetElement("calendarOverridesInfo");
Element cdrOverrides = cdrOverridesInfo.GetElement("calendarOverrides");
cdrOverrides.AppendValue("US");
cdrOverrides.AppendValue("JN");
cdrOverridesInfo.SetElement("calendarOverridesOperation", "CDR_AND");

**NOTE**: "calendarOverridesOperation" can be omitted only if one "calendarOverrides" is specified.

**Overrides**: Append overrides to modify the calculation.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldID</td>
<td></td>
<td>string</td>
<td>Specify a field mnemonic or alpha-numeric, such as PR092 or PRICING_SOURCE. Review FLDs&lt;GO&gt; for list of possible overrides.</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td>string</td>
<td>The desired override value</td>
</tr>
</tbody>
</table>

**Example Syntax**: Element overrides = request["overrides"];
Element override1 = overrides.AppendElement();
override1.SetElement("fieldId", "BEST_DATA_SOURCE_OVERRIDE");
override1.SetElement("value", "BLI");
A.2.5 HistoricalDataResponse: Choice

Figure A-2 provides the structure of a Historical Data Response. See “Reference Data Service Response” on page 182 for more information.
A.2.6 IntradayTickRequest: Sequence

<table>
<thead>
<tr>
<th>Securities: A stock or bond.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>securities</td>
</tr>
</tbody>
</table>

Example Syntax: `Element securities = request.GetElement("securities"); request.Set("security", "VOD LN Equity");`

**Start Date:** the first date of the period to retrieve data

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>startDateTime</td>
<td>yyyy-mm-dd Thh:mm:ss</td>
<td>string</td>
<td>The start date and time.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("startDateTime", "2010-04-27T15:55:00");`

**End Date:** the end date of the period to retrieve data

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>endDateTime</td>
<td>yyyy-mm-dd Thh:mm:ss</td>
<td>string</td>
<td>The end date and time.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("endDateTime", "2010-04-27T16:00:00");`

**Event Type:** The requested data event type

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>eventType</td>
<td>TRADE</td>
<td>string</td>
<td>Corresponds to LAST_PRICE</td>
</tr>
<tr>
<td></td>
<td>BID</td>
<td>string</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>ASK</td>
<td>string</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
<tr>
<td></td>
<td>BID_BEST</td>
<td>string</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>ASK_BEST</td>
<td>string</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
<tr>
<td></td>
<td>MID_PRICE</td>
<td>string</td>
<td>MID_PRICE only applies to the LSE. The mid price is equal to the sum of the best bid price and the best offer price divided by two, and rounded up to be consistent with the relevant price format.</td>
</tr>
<tr>
<td></td>
<td>AT_TRADE</td>
<td>string</td>
<td>Automatic trade for London Sets stocks.</td>
</tr>
<tr>
<td></td>
<td>BEST_BID</td>
<td>string</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>BEST_ASK</td>
<td>string</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("eventType", "TRADE");`
Include Condition Codes: return any condition codes that may be associated to a tick, which identifies extraordinary trading and quoting circumstances.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeConditionCodes</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>A comma delimited list of exchange condition codes associated with the event. Review QR&lt;GO&gt; for more information on each code returned.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeConditionCodes", true);`

Include Non Plottable Events: return ticks in the response that have condition codes

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeNonPlottableEvents</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Returns all ticks, including those with condition codes.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeNonPlottableEvents", true);`

Include Exchange Codes: return the exchange code of the trade

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeExchangeCodes</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>The exchange code where this tick originated. Review QR&lt;GO&gt; for more information.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeExchangeCodes", true);`

Return Entitlements: returns the entitlement identifiers associated with security.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnEids</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Option on whether to return EIDs for the security.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("returnEids", true);`

Include Broker Codes: return the broker code of the trade

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeBrokerCodes</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>The broker code for Canadian, Finnish, Mexican, Philippine, and Swedish equities only. The Market Maker Lookup screen, MMTK&lt;GO&gt;, displays further information on market makers and their corresponding codes.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeBrokerCodes", true);`

Include Reporting Party Side Codes: return transaction codes

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeRpsCodes</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>The Reporting Party Side. The following values appear: -B: A customer transaction where the dealer purchases securities from the customer. -S: A customer transaction where the dealer sells securities to the customer. -D: An inter-dealer transaction (always from the sell side).</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeRpsCodes", true);`
A.2.7 IntradayTickResponse: Choice

Figure A-3 provides the structure of an Intraday Tick Response. See “Reference Data Service Response” on page 182 for more information.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeBicMicCodes</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>The BIC, or Bank Identifier Code, as a 4-character unique identifier for each bank that executed and reported the OTC trade, as required by MiFID. BICs are assigned and maintained by SWIFT (Society for Worldwide Interbank Financial Telecommunication). The MIC is the Market Identifier Code, and this indicates the venue on which the trade was executed.</td>
</tr>
</tbody>
</table>

Example Syntax: `request.Set("includeBicMicCodes", true);`
Figure A-3: IntradayTickResponse
### A.2.8 IntradayBarRequest: Sequence

**Securities:** A stock or bond.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>securities</td>
<td>string</td>
<td></td>
<td>See &quot;Security/Securities&quot; on page 77 for additional details.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
Element securities = request.GetElement("securities");
request.Set("security", "VOD LN Equity");
```

**Start Date:** the first date of the period to retrieve data

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDate</td>
<td>yyyy-mm-dd Thh:mm:ss</td>
<td>string</td>
<td>The start date and time.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
request.Set("startDate", "2010-04-27T15:55:00");
```

**End Date:** the end date of the period to retrieve data

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endDate</td>
<td>yyyy-mm-dd Thh:mm:ss</td>
<td>string</td>
<td>The end date and time.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
request.Set("endDate", "2010-04-27T16:00:00");
```

**Event Type:** The requested data event type

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventType</td>
<td>TRADE string</td>
<td>TRADE</td>
<td>Corresponds to LAST_PRICE</td>
</tr>
<tr>
<td></td>
<td>BID string</td>
<td>BID</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>ASK string</td>
<td>ASK</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
<tr>
<td></td>
<td>BID_BEST string</td>
<td>BID_BEST</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>ASK_BEST string</td>
<td>ASK_BEST</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
<tr>
<td></td>
<td>BEST_BID string</td>
<td>BEST_BID</td>
<td>Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.</td>
</tr>
<tr>
<td></td>
<td>BEST_ASK string</td>
<td>BEST_ASK</td>
<td>Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
request.Set("eventType", "TRADE");
```

**Interval:** the length of each bar returned

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>1...1440 integer</td>
<td></td>
<td>Sets the length of each time bar in the response. Entered as a whole number, between 1 and 1440 in minutes. If omitted, the request will default to one minute. One minute is the lowest possible granularity.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
request.Set("interval", 60);
```
| **Gap Fill Initial Bar:** populate an empty bar with previous value |
| --- | --- | --- | --- |
| **Element** | **Element Value** | **Type** | **Description** |
| gapFillInitialBar | TRUE or FALSE | Boolean | When set to true, a bar contains the previous bar values if there was no tick during this time interval. |

**Example Syntax:** request.Set("gapFillInitialBar", true);

**Return Entitlements:** returns the entitlement identifiers associated with security.

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>returnEids</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Option on whether to return EIDs for the security.</td>
</tr>
</tbody>
</table>

**Example Syntax:** request.Set("returnEids", true);

**Adjustment Normal:** Adjust "change on day"

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentNormal</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing to reflect: Regular Cash, Interim, 1st Interim, 2nd Interim, 3rd Interim, 4th Interim, 5th Interim, Income, Estimated, Partnership Distribution, Final, Interest on Capital, Distribution, Prorated.</td>
</tr>
</tbody>
</table>

**Example Syntax:** request.Set("adjustmentNormal", true);

**Adjustment Abnormal:** Adjust for Abnormal Cash Dividends

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentAbnormal</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing to reflect: Special Cash, Liquidation, Capital Gains, Long-Term Capital Gains, Short-Term Capital Gains, Memorial, Return of Capital, Rights Redemption, Miscellaneous, Return Premium, Preferred Rights Redemption, Proceeds/Rights, Proceeds/Shares, Proceeds/Warrants.</td>
</tr>
</tbody>
</table>

**Example Syntax:** request.Set("adjustmentAbnormal", true);

**Adjustment Split:** Capital Changes Defaults

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentSplit</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Adjust historical pricing and/or volume to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, Rights Offerings/Entitlement.</td>
</tr>
</tbody>
</table>

**Example Syntax:** request.Set("adjustmentSplit", true);

**Adjustment Follow DPDF:** Follow the BLOOMBERG PROFESSIONAL service function DPDF<GO>

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Element Value</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>adjustmentFollowDPDF</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Setting to true will follow the DPDF&lt;GO&gt; BLOOMBERG PROFESSIONAL service function. True is the default setting for this option.</td>
</tr>
</tbody>
</table>

**Example Syntax:** request.Set("adjustmentFollowDPDF", true);
A.2.9 IntradayBarResponse: Choice

Figure A-4 provides the structure of an Intraday Bar Response. See “Reference Data Service Response” on page 182 for more information.

Figure A-4: IntradayBarResponse
### A.2.10 PortfolioDataRequest: Sequence

<table>
<thead>
<tr>
<th>Securities: A Portfolio ID</th>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>securities</td>
<td>string array</td>
<td>string</td>
<td>The users portfolio is identified by its Portfolio ID, which can be found on the upper right hand corner of the settings tab on the portfolio's PRTU&lt;GO&gt; page on the BLOOMBERG PROFESSIONAL service.</td>
<td></td>
</tr>
</tbody>
</table>

Example Syntax:
```csharp
Element securities = request.GetElement("securities");
securities.AppendValue("UXXXXXXX-X Client");
```

<table>
<thead>
<tr>
<th>Fields: The desired reference fields.</th>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>string</td>
<td>The fields that can be used are PORTFOLIO_MEMBER, PORTFOLIO_MPOSITION, PORTFOLIO_MWEIGHT &amp; PORTFOLIO_DATA.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Syntax:
```csharp
Element fields = request.GetElement("fields");
fields.AppendValue("PORTFOLIO_MEMBER");
```

<table>
<thead>
<tr>
<th>Overrides: The Portfolio information can also be accessed historically by using the REFERENCE_DATE override field by supplying the date in 'yyymmmdd' format.</th>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldId</td>
<td>string</td>
<td>Field mnemonic &quot;REFERENCE_DATE&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The date in 'yyymmmdd' format.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Syntax:
```csharp
Element overrides = request["overrides"];
Element override1 = overrides.AppendElement();
override1.SetElement("fieldId", "REFERENCE_DATE");
override1.SetElement("value", "20100111");
```
A.2.11 PortfolioDataResponse: Choice

Figure A-5 provides the structure of a PortfolioDataResponse. See “Reference Data Service Response” on page 182 for more information.
### A.2.12 BEQSRequest: Sequence

**screenName**: An EQS screen name

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenName</td>
<td>string</td>
<td>string</td>
<td>(Required) The name of the screen to execute. It can be a user defined EQS screen or one of the Bloomberg Example screens on EQS &lt;GO&gt; on the BLOOMBERG PROFESSIONAL service.</td>
</tr>
</tbody>
</table>

**Example Syntax**: `request.Set("screenName", "Global Volume Surges");`

**screenType**: Screen Type.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>screenType</td>
<td>PRIVATE or GLOBAL</td>
<td>string</td>
<td>Use PRIVATE for user-defined EQS screen. Use GLOBAL for Bloomberg EQS screen.</td>
</tr>
</tbody>
</table>

**Example Syntax**: `request.Set("screenType", "GLOBAL");`

**languageId**: Specify the language for field names to be returned for screen data

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>languageId</td>
<td>string</td>
<td>string</td>
<td>The following languages are supported: ENGLISH, KANJI, FRENCH, GERMAN, SPANISH, PORTUGUESE, ITALIAN, CHINESE_TRA, KOREAN, CHINESE_SIM, THAI, SWED, FINNISH, DUTCH, MALAY, RUSSIAN, GREEK, POLISH, DANISH, FLEMISH, ESTONIAN, TURKISH, NORWEGIAN, LATVIAN, LITHUANIAN, INDONESIAN</td>
</tr>
</tbody>
</table>

**Example Syntax**: `request.Set("languageId", "FRENCH");`

**Group**: Specify group name.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>string</td>
<td>string</td>
<td>Screen folder name here as defined in EQS&lt;GO&gt;.</td>
</tr>
</tbody>
</table>

**Example Syntax**: `request.Set("Group", "Global Emerging Markets");`

**Overrides**: The EQS information can also be accessed historically by using the PitDate override field and supplying the date in 'yyyymmdd' format.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldId</td>
<td>string</td>
<td>string</td>
<td>Field mnemonic &quot;PitDate&quot;</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The date in 'yyyymmdd' format.</td>
<td></td>
</tr>
</tbody>
</table>

**Example Syntax**: ```
Element overrides = request.getElement("overrides");
Element override1 = overrides.appendElement();
override1.setElement("fieldId", "PitDate");
override1.setElement("value", "20121210");```
A.2.13 **BEQSResponse: Choice**

Figure A-1 provides the structure of a BEQSResponse. See “Reference Data Service Response” on page 182 for more information.
A.2.14 Reference Data Service Response

Table A-1 and Table A-2 provides descriptions of the individual elements received in a reference data response. Please view pages 164, 170, 174, 177, and 181 for information on the structure of each response.

Table A-1: Reference Data Service Response Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseError</td>
<td>Returned when a request cannot be completed for any reason. It is an errorInfo element.</td>
</tr>
<tr>
<td>securityData[ ]</td>
<td>Contains an array of securityData elements</td>
</tr>
<tr>
<td>securityData</td>
<td>Contains the response data for a specific security from a ReferenceDataRequest or a HistoricalDataRequest. It provides the security string specified in the request, the sequence number and can include fieldData[ ], fieldsExceptions[ ] and securityError elements.</td>
</tr>
<tr>
<td>barData</td>
<td>Contains the response data for an IntradayBarRequest. It can provide a barTickData[ ] element and/or an eidData array element.</td>
</tr>
<tr>
<td>barTickData[ ]</td>
<td>Contains an array of barTickData elements</td>
</tr>
<tr>
<td>barTickData</td>
<td>Contains values associated to the bar, including time, open, high, low, close, volume, numEvents.</td>
</tr>
<tr>
<td>tickData</td>
<td>Contains the response data for an IntradayTickRequest. It can provide a tickData[ ] element and/or an eidData array element.</td>
</tr>
<tr>
<td>tickData[ ]</td>
<td>Contains an array of tickData elements</td>
</tr>
<tr>
<td>tickData[ ] ::  tickData</td>
<td>Contains values associated to the eventType, including time, type, value, size, condition code, and exchange code.</td>
</tr>
<tr>
<td>eidData[ ]</td>
<td>Contains a list of eidData values associated to the securities requested. If the requestor does not have the entitlement as per EXCH&lt;GO&gt; then the identifiers will not be returned.</td>
</tr>
<tr>
<td>securityError</td>
<td>Returned when a request cannot be completed for any reason. It is an errorInfo element.</td>
</tr>
<tr>
<td>fieldExceptions[ ]</td>
<td>Contains an array of fieldExceptions.</td>
</tr>
<tr>
<td>fieldExceptions</td>
<td>Contains a field identifier, message and errorInfo element.</td>
</tr>
<tr>
<td>fieldData[ ]</td>
<td>Contains an array of fieldData values</td>
</tr>
<tr>
<td>fieldData</td>
<td><strong>Reference Data Request:</strong> element with the fieldId and value</td>
</tr>
<tr>
<td></td>
<td><strong>Historical Data Request:</strong> element with the relativeDate, Date, fieldId and value</td>
</tr>
<tr>
<td>errorInfo</td>
<td>Contains values about the error which has occurred, including the source, code, category, message, and subcategory.</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
</tr>
<tr>
<td>security</td>
<td>String</td>
</tr>
<tr>
<td>eidData</td>
<td>Integer</td>
</tr>
<tr>
<td>sequenceNumber</td>
<td>Integer</td>
</tr>
<tr>
<td>fieldId</td>
<td>String</td>
</tr>
<tr>
<td>relativeDate</td>
<td>String</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Time</td>
<td>DateTime</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Value</td>
<td>Integer</td>
</tr>
<tr>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Datetime</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>Integer</td>
</tr>
<tr>
<td>conditionCode</td>
<td>String</td>
</tr>
<tr>
<td>exchangeCode</td>
<td>String</td>
</tr>
<tr>
<td>Source</td>
<td>String</td>
</tr>
<tr>
<td>Code</td>
<td>Integer</td>
</tr>
<tr>
<td>Category</td>
<td>String</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
</tr>
<tr>
<td>subcategory</td>
<td>String</td>
</tr>
</tbody>
</table>
A  Schemas 184

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpsCode</td>
<td>String</td>
<td>Transaction code. The following values appear: -B: A customer transaction where the dealer purchases securities from the customer. -S: A customer transaction where the dealer sells securities to the customer. -D: An inter-dealer transaction (always from the sell side).</td>
</tr>
<tr>
<td>brokerBuyCode</td>
<td>String</td>
<td>The broker code for Canadian, Finnish, Mexican, Philippine, and Swedish equities only. The Market Maker Lookup screen, MMTK on the BLOOMBERG PROFESSIONAL service, displays further information on market makers and their corresponding codes. To display the broker's name, enter: MMID {market maker code} &lt;GO&gt;.</td>
</tr>
<tr>
<td>brokerSellCode</td>
<td>String</td>
<td>The MIC is the Market Identifier Code, and this indicates the venue on which the trade was executed.</td>
</tr>
<tr>
<td>micCode</td>
<td>String</td>
<td>The BIC, or Bank Identifier Code, as a 4-character unique identifier for each bank that executed and reported the OTC trade, as required by MiFID. BICs are assigned and maintained by SWIFT (Society for Worldwide Interbank Financial Telecommunication).</td>
</tr>
</tbody>
</table>
A.3 Schema for API Field Service //blp//apiflds

A.3.1 Requests: Choice

Top level request to the service.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfoRequest</td>
<td>FieldInfoRequest</td>
<td>Request for field information.</td>
</tr>
<tr>
<td>fieldSearchRequest</td>
<td>FieldSearchRequest</td>
<td>Field search information.</td>
</tr>
<tr>
<td>categorizedFieldSearchRequest</td>
<td>CategorizedFieldSearchRequest</td>
<td>See “Categorized Field Search Request” on page 191.</td>
</tr>
</tbody>
</table>

A.3.2 Responses: Choice

Top level request to the service.

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldResponse</td>
<td>FieldResponse</td>
<td>Field response information.</td>
</tr>
<tr>
<td>categorizedFieldResponse</td>
<td>CategorizedFieldResponse</td>
<td>See “Categorized Field Search Request Response” on page 192.</td>
</tr>
</tbody>
</table>

A.3.3 Field Information Request

<table>
<thead>
<tr>
<th>Identifier: the reference or streaming fields desired.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>

Example Syntax:
```csharp
Element idList = request.GetElement("id");
request.Append("id", "LAST_PRICE");
request.Append("id", "pq005");
```

Return field documentation:

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnFieldDocumentation</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>Returns a description about the field as seen on FLDS&lt;GO&gt;. Default value is false.</td>
</tr>
</tbody>
</table>

Example Syntax:
```csharp
request.Set("returnFieldDocumentation", true);
```
A.3.3.1 Field Information Request Response

See “Field Service Response Elements” on page 196 and “Field Service Response Values” on page 197 for more information.
## A.3.4 Field Search Request

**Identifier:** the reference or streaming fields desired.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchSpec</td>
<td>String</td>
<td></td>
<td>The string argument to search through mnemonics, descriptions and definitions. It is also able to 'intelligently' expand works, i.e. mkt ==&gt; market.</td>
</tr>
</tbody>
</table>

```
Example Syntax: request.Set("searchSpec", "mutual fund");
```

### Include options:

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>New Fields Analysis Corporate Actions Custom Fields Descriptive Earnings Estimates Fundamentals Market Activity Metadata Ratings Trading Systems</td>
<td>String</td>
<td>Categories for fields</td>
</tr>
<tr>
<td>productType</td>
<td>All Govt Corp Mtge M-Mkt Muni Pfd Equity Cmdty Index Curncy</td>
<td>String</td>
<td>The results will be filtered by fields that are available for this yellow key (security type).</td>
</tr>
</tbody>
</table>
### fieldType

<table>
<thead>
<tr>
<th>fieldType</th>
<th>String</th>
<th>Results include fields that are both streaming (real-time and delayed) and reference (static)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realtime</td>
<td>String</td>
<td>Results include fields that provide streaming data (real-time and delayed)</td>
</tr>
<tr>
<td>Static</td>
<td>String</td>
<td>Results include fields that provide reference data (static).</td>
</tr>
</tbody>
</table>

```java
Element element = request.getElement("include");
    element.setElement("productType", "Equity");
    element.setElement("fieldType", "Static");
Element element1 = element.GetElement("category");
    element1.AppendValue("Ratings");
    element1.AppendValue("Analysis");
```

### Exclude options:

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td></td>
<td></td>
<td>Categories for fields</td>
</tr>
<tr>
<td></td>
<td>New Fields</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corporate Actions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Custom Fields</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Descriptive</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Earnings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Estimates</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fundamentals</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Market Activity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metadata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ratings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trading</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>productType</td>
<td>All</td>
<td>String</td>
<td>The results will be filtered by fields that are available for this yellow key (security type).</td>
</tr>
<tr>
<td></td>
<td>Govt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Corp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mtge</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>M-Mkt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Muni</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pfd</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Equity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cmdty</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Curncy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.3.4.1 Field Search Request Response

See “Field Service Response Elements” on page 196 and “Field Service Response Values” on page 197 for more information.
Figure A-7: Field Search Request Response
### A.3.5 Categorized Field Search Request

**Identifier:** the reference or streaming fields desired.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchSpec</td>
<td></td>
<td>String</td>
<td>The string argument to search through mnemonics, descriptions and definitions. It is also able to 'intelligently' expand works, i.e. mkt ==&gt; market.</td>
</tr>
</tbody>
</table>

Example Syntax: request.Set("searchSpec", "mutual fund");

**Exclude options:**

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>New Fields, Analysis, Corporate Actions, Custom Fields, Descriptive, Earnings, Estimates, Fundamentals, Market Activity, Metadata, Ratings, Trading, Systems</td>
<td>String</td>
<td>Categories for fields</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>productType</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td></td>
<td>String</td>
<td>The results will be filtered by fields that are available for this yellow key (security type).</td>
</tr>
<tr>
<td>Govt</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Corp</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Mtge</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>M-Mkt</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Muni</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Pfđ</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Equity</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Cmdty</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Curncy</td>
<td></td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
A.3.5.1 Categorized Field Search Request Response

See “Field Service Response Elements” on page 196 and “Field Service Response Values” on page 197 for more information.
A.3.6 Field List Request

<table>
<thead>
<tr>
<th>Identifier: the reference or streaming fields desired.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>fieldType</td>
</tr>
<tr>
<td>Realtime</td>
</tr>
<tr>
<td>Static</td>
</tr>
</tbody>
</table>

Example Syntax: element.setElement("fieldType", "Static");

Return field documentation:

| Element | **Element Value** | **Type** | **Description** |
|-----------------------------------------------|
| returnFieldDocumentation | TRUE or FALSE | Boolean | Returns a description about the field as seen on FLDS<GO>. Default value is false. |

Example Syntax: request.Set("returnFieldDocumentation", true);

A.3.6.1 Field List Request Response

See “Field Service Response Elements” on page 196 and “Field Service Response Values” on page 197 for more information.
Figure A-9: Field List Request Response
### A.3.7 Field Service Response Elements

The following table provides descriptions of the individual elements received in the field service responses. Please view graphs A.3.3, A.3.5, A.3.7 and A.3.9 for information on the structure of the response.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldSearchError</td>
<td>Returned when a request cannot be completed for any reason. It is an errorInfo element.</td>
</tr>
<tr>
<td>fieldData[]</td>
<td>Contains an array of fieldData values</td>
</tr>
<tr>
<td>fieldData</td>
<td>Contains a id corresponding to the requested field identifier, along with either a fieldInfo or fieldError element</td>
</tr>
<tr>
<td>fieldInfo</td>
<td>Contains values on the mnemonic, datatype, categoryName, description, and documentation.</td>
</tr>
<tr>
<td>fieldError</td>
<td>Returned when a request cannot be completed for any reason or in the case of a fieldInfoRequest when an invalid field mnemonic or alphanumeric is entered.</td>
</tr>
<tr>
<td>categorizedFieldSearchError</td>
<td>Returned when a request cannot be completed for any reason. It is an errorInfo element.</td>
</tr>
<tr>
<td>category[]</td>
<td>Contains an array of category elements.</td>
</tr>
<tr>
<td>category</td>
<td>Contains categoryName, categoryId, numFields, descriptions, isLeafNode and a fieldData[] element.</td>
</tr>
<tr>
<td>errorInfo</td>
<td>Contains values about the error which has occurred, including the source, code, category, message, and subcategory.</td>
</tr>
</tbody>
</table>
### A.3.8 Field Service Response Values

<table>
<thead>
<tr>
<th><strong>Element</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Resulting field represented as an alphanumeric or a Mnemonic, i.e., PR005 or PX_LAST.</td>
</tr>
<tr>
<td>mnemonic</td>
<td>Integer</td>
<td>Resulting field represented as a mnemonic, i.e., PX_LAST.</td>
</tr>
<tr>
<td>datatype</td>
<td>Enumeration</td>
<td>Enumeration values representing Bloomberg data types. Please see specific SDK documentation for the enum values.</td>
</tr>
<tr>
<td>ftype</td>
<td>Enumeration</td>
<td>Enumeration value representing data types shown in XDM&lt;GO&gt;.</td>
</tr>
<tr>
<td>categoryName</td>
<td>String</td>
<td>Response value for the name of the category. Could be one of the following: New Fields, Analysis, Corporate Actions, Custom Fields, Descriptive, Earnings Estimates, Fundamentals, Market Activity, Metadata, Ratings, and Trading Systems.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Is the short description describing the field, for example for the mnemonic LAST_PRICE the description is &quot;Last Trade/Last Price&quot;.</td>
</tr>
<tr>
<td>documentation</td>
<td>String</td>
<td>Corresponds to the definition in FLDS&lt;GO&gt;.</td>
</tr>
<tr>
<td>Time</td>
<td>DateTime</td>
<td>Tick time for an intraday tick request</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td>The event type for an intraday tick</td>
</tr>
<tr>
<td>Source</td>
<td>String</td>
<td>Bloomberg internal error source information.</td>
</tr>
<tr>
<td>Code</td>
<td>Integer</td>
<td>Bloomberg internal error code.</td>
</tr>
<tr>
<td>Category</td>
<td>String</td>
<td>Bloomberg error classification. Used to determine the general classification of the failure.</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Human readable description of the failure.</td>
</tr>
<tr>
<td>subcategory</td>
<td>String</td>
<td>Bloomberg sub-error classification. Used to determine the specific classification of the failure.</td>
</tr>
</tbody>
</table>
A.4 Market Bar Subscription

A.4.1 Market Bar Subscription Settings

<table>
<thead>
<tr>
<th>Argument Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Security       | string | As with any Subscription, a Market Bar Subscription must contain at least one security, field and Correlation ID. The topic is defined as: 
"//blp/mktbar/symbology/identifier" |
| Fields         | string | MKTBAR service is based on TRADE ticks only. Hence, subscription topic string should have option "fields=LAST_PRICE". See “Fields” on page 79 for additional details. Fields can be specified as a alpha numeric or mnemonic. |
| bar_size       | string | Length of the bar defined in minutes. The minimum supported size of the bar is 1 min. The maximum supported size of the bar is 1440 minutes, (=24 hours). |
| start_time     | string | Optional. This should be in the format hh:mm. If not set then the time of session start of the security or subscription time will be used. |
| end_time       | string | Optional. This should be in the format hh:mm. If not specified then session end time of the security will be used. |

Example Syntax:

```java
Subscription mySubscription = new Subscription("//blp/mktbar/TICKERX/IBM US Equity",
"last_price",
"bar_size=5&start_time=13:30&end_time=20:00",
new CorrelationID("IBM US Equity"));
```

A.4.2 Market Bar Subscription: Data Events Response

Each bar update will include two time fields TIME and DATE_TIME. Both time fields will be of datetime type. While TIME carries the time of the current bar DATE_TIME will also include the date of the bar thereby indicating the date change if subscription left running overnight.
MarketBarStart

/blp/mktbar/TICKER/IBM US Equity - MarketBarStart
  TIME = 12:5
  OPEN = 176.88
  HIGH = 176.89
  LOW = 176.85
  CLOSE = 176.88
  NUMBER_OF_TICKS = 12
  VOLUME = 1400
  VALUE = 247622.0
  DATE_TIME = 2/7/2014 12:5

MarketBarUpdate

//blp/mktbar/TICKER/IBM US Equity - MarketBarUpdate
  TIME = 12:5
  HIGH = 176.89
  LOW = 176.85
  CLOSE = 176.87
  NUMBER_OF_TICKS = 13
  VOLUME = 1500
  VALUE = 265309.0
  DATE_TIME = 2/7/2014 12:5

MarketBarIntervalEnd

//blp/mktbar/TICKER/IBM US Equity - MarketBarIntervalEnd
  TIME = 12:5
  DATE_TIME = 2/7/2014 12:5

MarketBarEnd

//blp/mktbar/TICKER/IBM US Equity - MarketBarEnd
  TIME = 12:5
  DATE_TIME = 2/7/2014 12:5
<table>
<thead>
<tr>
<th>Argument Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>datetime</td>
<td>Returns the time of the start of bar bucket.</td>
</tr>
<tr>
<td>OPEN</td>
<td>Float64</td>
<td>Returns open price of the bar bucket. Should be returned in the MarketBarStart event.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Float64</td>
<td>Returns high price of the bar bucket in the MktBarStart and subsequently in every MktBarUpdate if higher price occurs until the end of the bar.</td>
</tr>
<tr>
<td>LOW</td>
<td>Float64</td>
<td>Returns low price of the bar bucket in the MktBarStart and subsequently in every MktBarUpdate if lower price occurs until the end of the bar.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Float64</td>
<td>Returns updated close price on every in MktBarStart and MktBarUpdate event.</td>
</tr>
<tr>
<td>NUMBER_OF_TICKS</td>
<td>Int32</td>
<td>Accumulates number of ticks in the bar on every MktBarStart and MktBarUpdate event till MarketBarIntervalEnd is sent.</td>
</tr>
<tr>
<td>VALUE</td>
<td>Float64</td>
<td>Volume*Price increments for number of trades in each market bar and is reset at the start of each market bar.</td>
</tr>
<tr>
<td>VOLUME</td>
<td>Int64</td>
<td>Volume increments for number of trades in each market bar and is reset at the start of each market bar.</td>
</tr>
<tr>
<td>DATE_TIME</td>
<td>datetime</td>
<td>Returns the date and time of the bar bucket  NOTE: value of the field consists of MM/DD/YYYY HH:MM.</td>
</tr>
</tbody>
</table>

**Example Syntax:**
- `datetime time = msg.getElementAsDatetime(TIME);`
- `int open = msg.getElementAsFloat64(OPEN);`
- `int high = msg.getElementAsFloat64(HIGH);`
- `int low = msg.getElementAsFloat64(LOW);`
- `int close = msg.getElementAsFloat64(CLOSE);`
- `int number_of_ticks = msg.getElementAsInt32(NUMBER_OF_TICKS);`
- `float value = msg.getElementAsInt64(VALUE);`
- `float volume = msg.getElementAsInt64(VOLUME);`
- `Datetime datetime = msg.getElementAsDatetime(DATE_TIME);`
A.5 Schema for Market Data and Custom VWAP

A.5.1 MarketDataEvents: Choice

Events related to Market Data:

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketDataEvents</td>
<td>MarketDataEvents</td>
<td>Market Data Events</td>
</tr>
</tbody>
</table>

A.5.2 Market Data Service Subscription Options

<table>
<thead>
<tr>
<th>Argument Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval</td>
<td>string</td>
<td>Sets a defined period in seconds for which updates will be received for the subscription. The range for this argument is 0.10 to 86400.00, which is equal to 100ms to 24hours. For example setting this argument to 30 will result in the requesting application to receive updates every 30 seconds for the requested securities.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
Subscription mySubscription = new Subscription(security, fields, "interval=30.0", new CorrelationID(security));
```

<table>
<thead>
<tr>
<th>Argument Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delayed</td>
<td>string</td>
<td>Forces the subscription to be delayed even if the requestor has real-time exchange entitlements.</td>
</tr>
</tbody>
</table>

Example Syntax:
```
Subscription mySubscription = new Subscription(security, fields, "delayed", new CorrelationID(security));
```

A.5.3 MarketDataEvents: Sequence

Fields in subscription

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TORONTO_MOC_ELIGIBLE_REALTIME</td>
<td>Optional Boolean</td>
<td>Toronto MOC Eligible</td>
</tr>
<tr>
<td>NASDAQ_CLOSING_CROSS_ELIGIBLE_RT</td>
<td>Optional Boolean</td>
<td>Nasdaq Closing Cross Eligible</td>
</tr>
<tr>
<td>MGF_SETTING_RT</td>
<td>Optional Boolean</td>
<td>MGF Setting (Real-time)</td>
</tr>
<tr>
<td>RT_EXCH_TRADE_STATUS</td>
<td>Optional Boolean</td>
<td>Exchange Trading Status</td>
</tr>
<tr>
<td>RT_QUOTE_STATUS</td>
<td>Optional Boolean</td>
<td>Quotation Status</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>IND_BID_FLAG</td>
<td>Optional Boolean</td>
<td>Indicative Bid Flag</td>
</tr>
<tr>
<td>IND_ASK_FLAG</td>
<td>Optional Boolean</td>
<td>Indicative Ask Flag</td>
</tr>
<tr>
<td>TRADING_DT_REALTIME</td>
<td>Optional Date</td>
<td>Trading Date</td>
</tr>
<tr>
<td>RT_TIME_OF_TRADE</td>
<td>Optional Datetime</td>
<td>Time Trade Occurred</td>
</tr>
<tr>
<td>CR_OBSERVATION_DATE</td>
<td>Optional Datetime</td>
<td>Current Observation Date</td>
</tr>
<tr>
<td>PRIOR_OBSERVATION_DATE</td>
<td>Optional Datetime</td>
<td>Prior Observation Date</td>
</tr>
<tr>
<td>TIME</td>
<td>Optional Datetime</td>
<td>Time of Last Update</td>
</tr>
<tr>
<td>VOLUME</td>
<td>Optional Int32</td>
<td>Volume</td>
</tr>
<tr>
<td>BID_YIELD</td>
<td>Optional Float32</td>
<td>Bid Yield</td>
</tr>
<tr>
<td>ASK_YIELD</td>
<td>Optional Float32</td>
<td>Ask Yield</td>
</tr>
<tr>
<td>RT_OPEN_INTEREST</td>
<td>Optional Float32</td>
<td>Open Interest (Real-time)</td>
</tr>
<tr>
<td>OFF_ON_EXCH_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Off And On Exchange Volume (Real-time)</td>
</tr>
<tr>
<td>OFF_EXCH_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Off Exchange Volume (Real-time)</td>
</tr>
<tr>
<td>PX_VOLUME_BAL_RT</td>
<td>Optional Int32</td>
<td>Volume Balance (Real-time)</td>
</tr>
<tr>
<td>DELTA_BID_RT</td>
<td>Optional Float32</td>
<td>Delta Bid (Real-time)</td>
</tr>
<tr>
<td>DELTA_ASK_RT</td>
<td>Optional Float32</td>
<td>Delta Ask (Real-time)</td>
</tr>
<tr>
<td>DELTA_MID_RT</td>
<td>Optional Float32</td>
<td>Delta Mid (Real-time)</td>
</tr>
<tr>
<td>DELTA_LAST_RT</td>
<td>Optional Float32</td>
<td>Delta Last Trade (Real-time)</td>
</tr>
<tr>
<td>GAMMA_BID_RT</td>
<td>Optional Float32</td>
<td>Gamma Bid (Real-time)</td>
</tr>
<tr>
<td>GAMMA_ASK_RT</td>
<td>Optional Float32</td>
<td>Gamma Ask (Real-time)</td>
</tr>
<tr>
<td>GAMMA_MID_RT</td>
<td>Optional Float32</td>
<td>Gamma Mid (Real-time)</td>
</tr>
<tr>
<td>GAMMA_LAST_RT</td>
<td>Optional Float32</td>
<td>Gamma Last Trade (Real-time)</td>
</tr>
<tr>
<td>VEGA_BID_RT</td>
<td>Optional Float32</td>
<td>Vega Bid (Real-time)</td>
</tr>
<tr>
<td>VEGA_ASK_RT</td>
<td>Optional Float32</td>
<td>Vega Ask (Real-time)</td>
</tr>
<tr>
<td>VEGA_MID_RT</td>
<td>Optional Float32</td>
<td>Vega Mid (Real-time)</td>
</tr>
<tr>
<td>VEGA_LAST_RT</td>
<td>Optional Float32</td>
<td>Vega Last Trade (Real-time)</td>
</tr>
<tr>
<td>IVOL_BID_RT</td>
<td>Optional Float32</td>
<td>Implied Volatility Bid (Real-time)</td>
</tr>
<tr>
<td>IVOL_ASK_RT</td>
<td>Optional Float32</td>
<td>Implied Volatility ASK (Real-time)</td>
</tr>
<tr>
<td>IVOL_MID_RT</td>
<td>Optional Float32</td>
<td>Implied Volatility Mid (Real-time)</td>
</tr>
<tr>
<td>IVOL_LAST_RT</td>
<td>Optional Float32</td>
<td>Implied Volatility Last Trade (Real-time)</td>
</tr>
<tr>
<td>EQY_SH_FOREIGN_RT</td>
<td>Optional Float32</td>
<td>Shares Available To Foreign Investors (Real-time)</td>
</tr>
<tr>
<td>LISTED_SH_RT</td>
<td>Optional Float32</td>
<td>Number Of Listed Shares (Real-time)</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BLP_SPRD_TO_BENCH_BID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Bid Spread To Benchmark (Real-time)</td>
</tr>
<tr>
<td>BLP_SPRD_TO_BENCH ASK_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Ask Spread To Benchmark (Real-time)</td>
</tr>
<tr>
<td>BLP_SPRD_TO_BENCH_MID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Mid Spread To Benchmark (Real-time)</td>
</tr>
<tr>
<td>BLP_Z_SPRD_MID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Mid Z Spread (Real-time)</td>
</tr>
<tr>
<td>BLP_ASW_SPREAD_MID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Mid ASW Spread (Real-time)</td>
</tr>
<tr>
<td>BLP_I_SPRD_MID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Mid I Spread (Real-time)</td>
</tr>
<tr>
<td>BLP_CDS_BASIS_MID_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Mid CDS Basis (Real-time)</td>
</tr>
<tr>
<td>BLP_SPRD_TO_BENCH_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Sprd To Bench Chg On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_Z_SPRD_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Z Spread Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_ASW_SPRD_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg ASW Spread Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_I_SPRD_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg I Spread Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_CDS_BASIS_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg CDS Basis Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_SPRD_TO_BENCH_PCT_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Spd To Bench % Chg On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_Z_SPRD_PCT_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg Z Spread % Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_ASW_SPRD_PCT_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg ASW Spread % Chg On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_I_SPRD_PCT_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg I Spread % Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_CDS_BASIS_PCT_CHG_RT</td>
<td>Optional Float32</td>
<td>Bloomberg CDS Basis % Change On Day (Real-time)</td>
</tr>
<tr>
<td>PX_SETTLE_ACTUAL_RT</td>
<td>Optional Float32</td>
<td>Settlement Price Actual (Real-time)</td>
</tr>
<tr>
<td>ARBITRAGE ASK ORD NOT MATCHED_RT</td>
<td>Optional Float32</td>
<td>Arbitrage Ask Orders Not Matched (Real-time)</td>
</tr>
<tr>
<td>ARBITRAGE BID ORD NOT MATCHED_RT</td>
<td>Optional Float32</td>
<td>Arbitrage Bid Orders Not Matched (Real-time)</td>
</tr>
<tr>
<td>NON ARBITRAGE ASK NOT MATCHED_RT</td>
<td>Optional Float32</td>
<td>Non Arbitrage Ask Orders Not Matched (Real-time)</td>
</tr>
<tr>
<td>NON ARBITRAGE BID NOT MATCHED_RT</td>
<td>Optional Float32</td>
<td>Non Arbitrage Bid Orders Not Matched (Real-time)</td>
</tr>
<tr>
<td>ARBITRAGE ASK ORD VOLUME_RT</td>
<td>Optional Int32</td>
<td>Arbitrage Ask Orders Volume (Real-time)</td>
</tr>
<tr>
<td>ARBITRAGE BID ORD VOLUME_RT</td>
<td>Optional Int32</td>
<td>Arbitrage Bid Orders Volume (Real-time)</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>NON_ARBIT_ASK_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Non Arbitrage Ask Orders Volume (Real-time)</td>
</tr>
<tr>
<td>NON_ARBIT_BID_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Non Arbitrage Bid Orders Volume (Real-time)</td>
</tr>
<tr>
<td>PRE_ANNOUNCE_NUM_PROG_ASK_RT</td>
<td>Optional Float32</td>
<td>Pre Announce Num of Program Ask Orders (Real-time)</td>
</tr>
<tr>
<td>PRE_ANNOUNCE_NUM_PROG_BID_RT</td>
<td>Optional Float32</td>
<td>Pre Announce Num of Program Bid Orders (Real-time)</td>
</tr>
<tr>
<td>TRUST_ASK_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Trust Ask Orders Volume (Real-time)</td>
</tr>
<tr>
<td>PROPRIETARY_ASK_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Proprietary Ask Orders Volume (Real-time)</td>
</tr>
<tr>
<td>TRUST_BID_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Trust Bid Orders Volume (Real-time)</td>
</tr>
<tr>
<td>PROPRIETARY_BID_ORD_VOLUME_RT</td>
<td>Optional Int32</td>
<td>Proprietary Bid Orders Volume (Real-time)</td>
</tr>
<tr>
<td>TOTAL_VOLUME_PROGRAM_TRADE_RT</td>
<td>Optional Int32</td>
<td>Total Volume of Program Trading (Real-time)</td>
</tr>
<tr>
<td>PX_INDICATIVE_BID_SIZE_RT</td>
<td>Optional Int32</td>
<td>Indicative Bid Price Size (Real-time)</td>
</tr>
<tr>
<td>PX_INDICATIVE_ASK_SIZE_RT</td>
<td>Optional Int32</td>
<td>Indicative Ask Price Size (Real-time)</td>
</tr>
<tr>
<td>NUM_TRADES_RT</td>
<td>Optional Int32</td>
<td>Number Of Trades</td>
</tr>
<tr>
<td>MGF_VOLUME_RT</td>
<td>Optional Int32</td>
<td>MGF Volume (Real-time)</td>
</tr>
<tr>
<td>NUM_TRADES_OPENING_AUCTION_RT</td>
<td>Optional Int32</td>
<td>Number Of Trades In Opening Auction (Real-time)</td>
</tr>
<tr>
<td>NUM_TRADES_CLOSING_AUCTION_RT</td>
<td>Optional Int32</td>
<td>Number Of Trades In Closing Auction (Real-time)</td>
</tr>
<tr>
<td>ALL_PRICE_SIZE</td>
<td>Optional Int32</td>
<td>All Price Size</td>
</tr>
<tr>
<td>RT_NYSE_LIQUIDITY_BID_SIZE</td>
<td>Optional Int32</td>
<td>NYSE Liquidity Quote Bid Size</td>
</tr>
<tr>
<td>RT_NYSE_LIQUIDITY_ASK_SIZE</td>
<td>Optional Int32</td>
<td>NYSE Liquidity Quote Ask Size</td>
</tr>
<tr>
<td>VOLUME_THEO</td>
<td>Optional Int32</td>
<td>Theoretical Volume</td>
</tr>
<tr>
<td>SIZE_LAST_AT_TRADE</td>
<td>Optional Int32</td>
<td>Size of Last AT Trade</td>
</tr>
<tr>
<td>SIZE_LAST_AT_TRADE_TDY</td>
<td>Optional Int32</td>
<td>Size of Today's Last AT Trade</td>
</tr>
<tr>
<td>OPEN_YLD</td>
<td>Optional Float32</td>
<td>Open Yield</td>
</tr>
<tr>
<td>OPEN_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's Open Yield</td>
</tr>
<tr>
<td>HIGH_YLD</td>
<td>Optional Float32</td>
<td>High Yield</td>
</tr>
<tr>
<td>HIGH_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's High Yield</td>
</tr>
<tr>
<td>LOW_YLD</td>
<td>Optional Float32</td>
<td>Low Yield</td>
</tr>
<tr>
<td>LOW_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's Low Yield</td>
</tr>
<tr>
<td>LAST_YLD</td>
<td>Optional Float32</td>
<td>Last Yield</td>
</tr>
<tr>
<td>LAST_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's Last Yield</td>
</tr>
<tr>
<td>SIZE_LAST_TRADE_TDY</td>
<td>Optional Int32</td>
<td>Size of Today's Last Trade</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>LAST2_YLD</td>
<td>Optional Float32</td>
<td>Last 2 Yield</td>
</tr>
<tr>
<td>LAST_DIR_YLD</td>
<td>Optional Int32</td>
<td>Last Yield Direction</td>
</tr>
<tr>
<td>LAST2_DIR_YLD</td>
<td>Optional Int32</td>
<td>Second Last Yield Direction</td>
</tr>
<tr>
<td>PREV_SES_LAST_YLD</td>
<td>Optional Float32</td>
<td>Previous Session Last Yield</td>
</tr>
<tr>
<td>BID2_YLD</td>
<td>Optional Float32</td>
<td>Bid 2 Yield</td>
</tr>
<tr>
<td>ASK2_YLD</td>
<td>Optional Float32</td>
<td>Ask 2 Yield</td>
</tr>
<tr>
<td>BID_DIR_YLD</td>
<td>Optional Int32</td>
<td>Bid Yield Direction</td>
</tr>
<tr>
<td>ASK_DIR_YLD</td>
<td>Optional Int32</td>
<td>Ask Yield Direction</td>
</tr>
<tr>
<td>MID_DIR</td>
<td>Optional Int32</td>
<td>Mid Direction</td>
</tr>
<tr>
<td>MID2_DIR</td>
<td>Optional Int32</td>
<td>Second Mid Direction</td>
</tr>
<tr>
<td>RT_PX_CHG_PCT_1D</td>
<td>Optional Float32</td>
<td>Real-Time Price Change 1 Day Percent</td>
</tr>
<tr>
<td>RT_YLD_CHG_NET_1D</td>
<td>Optional Float32</td>
<td>Real-Time Yield Change 1 Day Net</td>
</tr>
<tr>
<td>RT_YLD_CHG_PCT_1D</td>
<td>Optional Float32</td>
<td>Real-Time Yield Change 1 Day Percent</td>
</tr>
<tr>
<td>ASK_SIZE_TDY</td>
<td>Optional Int32</td>
<td>Today's Ask Size</td>
</tr>
<tr>
<td>BID_SIZE_TDY</td>
<td>Optional Int32</td>
<td>Today's Bid Size</td>
</tr>
<tr>
<td>VOLUME_TDY</td>
<td>Optional Int32</td>
<td>Today's Volume</td>
</tr>
<tr>
<td>BID_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's Bid Yield</td>
</tr>
<tr>
<td>ASK_YLD_TDY</td>
<td>Optional Float32</td>
<td>Today's Ask Yield</td>
</tr>
<tr>
<td>UP_LIMIT</td>
<td>Optional Float32</td>
<td>Up Limit</td>
</tr>
<tr>
<td>DOWN_LIMIT</td>
<td>Optional Float32</td>
<td>Down Limit</td>
</tr>
<tr>
<td>LAST_DIR</td>
<td>Optional Int32</td>
<td>Last Direction</td>
</tr>
<tr>
<td>LAST2_DIR</td>
<td>Optional Int32</td>
<td>Second Last Direction</td>
</tr>
<tr>
<td>BID_DIR</td>
<td>Optional Int32</td>
<td>Bid Direction</td>
</tr>
<tr>
<td>ASK_DIR</td>
<td>Optional Int32</td>
<td>Ask Direction</td>
</tr>
<tr>
<td>SIZE_LAST_TRADE</td>
<td>Optional Int32</td>
<td>Size of Last Trade</td>
</tr>
<tr>
<td>ASK_SIZE</td>
<td>Optional Int32</td>
<td>Ask Size</td>
</tr>
<tr>
<td>BID_SIZE</td>
<td>Optional Int32</td>
<td>Bid Size</td>
</tr>
<tr>
<td>LAST_PRICE</td>
<td>Optional Float64</td>
<td>Last Price</td>
</tr>
<tr>
<td>BID</td>
<td>Optional Float64</td>
<td>Bid Price</td>
</tr>
<tr>
<td>ASK</td>
<td>Optional Float64</td>
<td>Ask Price</td>
</tr>
<tr>
<td>HIGH</td>
<td>Optional Float64</td>
<td>High Price</td>
</tr>
<tr>
<td>LOW</td>
<td>Optional Float64</td>
<td>Low Price</td>
</tr>
<tr>
<td>BEST_BID</td>
<td>Optional Float64</td>
<td>Best Bid</td>
</tr>
<tr>
<td>BEST_ASK</td>
<td>Optional Float64</td>
<td>Best Ask</td>
</tr>
<tr>
<td>MID</td>
<td>Optional Float64</td>
<td>Mid Price</td>
</tr>
<tr>
<td>LAST_TRADE</td>
<td>Optional Float64</td>
<td>Last Trade</td>
</tr>
<tr>
<td>OPEN</td>
<td>Optional Float64</td>
<td>Open Price</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>PREV_SES_LAST_PRICE</td>
<td>Optional Float64</td>
<td>Previous Session Last Price</td>
</tr>
<tr>
<td>EXCH_VWAP</td>
<td>Optional Float64</td>
<td>Exchange VWAP</td>
</tr>
<tr>
<td>NASDAQ_OPEN</td>
<td>Optional Float64</td>
<td>NASDAQ Official Open Price</td>
</tr>
<tr>
<td>NASDAQ_FIRST_TRADE</td>
<td>Optional Float64</td>
<td>NASDAQ First Actual Trade</td>
</tr>
<tr>
<td>NASDAQ_PREV_BID</td>
<td>Optional Float64</td>
<td>NASDAQ Prevailing Bid Price</td>
</tr>
<tr>
<td>NASDAQ_PREV_ASK</td>
<td>Optional Float64</td>
<td>NASDAQ Prevailing Ask Price</td>
</tr>
<tr>
<td>INDICATIVE_FAR</td>
<td>Optional Float64</td>
<td>Far Indicative Price</td>
</tr>
<tr>
<td>INDICATIVE_NEAR</td>
<td>Optional Float64</td>
<td>Near Indicative Price</td>
</tr>
<tr>
<td>IMBALANCE_BID</td>
<td>Optional Float64</td>
<td>Net Order Imbalance Bid Price</td>
</tr>
<tr>
<td>IMBALANCE_ASK</td>
<td>Optional Float64</td>
<td>Net Order Imbalance Ask Price</td>
</tr>
<tr>
<td>ORDER_IMBBUY_VOLUME</td>
<td>Optional Int32</td>
<td>Net Order Imbalance Bid Volume</td>
</tr>
<tr>
<td>ORDER_IMBSELL_VOLUME</td>
<td>Optional Int32</td>
<td>Net Order Imbalance Ask Volume</td>
</tr>
<tr>
<td>VWAP</td>
<td>Optional Float64</td>
<td>Eqty intraday VWAP</td>
</tr>
<tr>
<td>FIXING_RATE_REALTIME</td>
<td>Optional Float64</td>
<td>Fixing Rate</td>
</tr>
<tr>
<td>HIGH_TEMP_REALTIME</td>
<td>Optional Float64</td>
<td>High Temperature</td>
</tr>
<tr>
<td>LOW_TEMP_REALTIME</td>
<td>Optional Float64</td>
<td>Low Temperature</td>
</tr>
<tr>
<td>MEAN_TEMP_REALTIME</td>
<td>Optional Float64</td>
<td>Mean Temperature</td>
</tr>
<tr>
<td>HEATING_DAYS_REALTIME</td>
<td>Optional Float64</td>
<td>Heating Degree Days</td>
</tr>
<tr>
<td>COOLING_DAYS_REALTIME</td>
<td>Optional Float64</td>
<td>Cooling Degree Days</td>
</tr>
<tr>
<td>REL_HUMIDITY_REALTIME</td>
<td>Optional Float64</td>
<td>Relative Humidity</td>
</tr>
<tr>
<td>WIND_SPEED_REALTIME</td>
<td>Optional Float64</td>
<td>Wind Speed</td>
</tr>
<tr>
<td>WEATHER_CODE_REALTIME</td>
<td>Optional Float64</td>
<td>Weather Condition Code</td>
</tr>
<tr>
<td>PRECIPITATION_REALTIME</td>
<td>Optional Float64</td>
<td>Precipitation</td>
</tr>
<tr>
<td>MARKET_DEFINED_VWAP_REALTIME</td>
<td>Optional Float64</td>
<td>Market Defined VWAP (Real-time)</td>
</tr>
<tr>
<td>MIN_LIMIT</td>
<td>Optional Float64</td>
<td>Minimum Limit Price</td>
</tr>
<tr>
<td>MAX_LIMIT</td>
<td>Optional Float64</td>
<td>Maximum Limit Price</td>
</tr>
<tr>
<td>THEO_PRICE</td>
<td>Optional Float64</td>
<td>Theoretical Price</td>
</tr>
<tr>
<td>MIN_LIMITOUT_OF_SESSION</td>
<td>Optional Float64</td>
<td>Minimum Limit Price Out Of Session</td>
</tr>
<tr>
<td>MAX_LIMITOUT_OF_SESSION</td>
<td>Optional Float64</td>
<td>Maximum Limit Price Out Of Session</td>
</tr>
<tr>
<td>BID_WEIGHTED_AVG_SPREAD</td>
<td>Optional Float64</td>
<td>Bid Weighted Average Spread</td>
</tr>
<tr>
<td>ASK_WEIGHTED_AVG_SPREAD</td>
<td>Optional Float64</td>
<td>Ask Weighted Average Spread</td>
</tr>
<tr>
<td>RT_NYSE LIQUIDITY_PX_BID</td>
<td>Optional Float64</td>
<td>NYSE Liquidity Quote Bid Price</td>
</tr>
<tr>
<td>RT_NYSE LIQUIDITY_PX_ASK</td>
<td>Optional Float64</td>
<td>NYSE Liquidity Quote Ask Price</td>
</tr>
<tr>
<td>INDICATIVE_BID</td>
<td>Optional Float64</td>
<td>Indicative Bid Price</td>
</tr>
<tr>
<td>INDICATIVE_ASK</td>
<td>Optional Float64</td>
<td>Indicative Ask Price</td>
</tr>
<tr>
<td>PX_EVAL_JAPANESE_REALTIME</td>
<td>Optional Float64</td>
<td>Japanese Evaluation Price</td>
</tr>
<tr>
<td>LAST_ALL_SESSIONS</td>
<td>Optional Float64</td>
<td>Last Price All Sessions</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>PX_NASDAQ_VWOP_REALTIME</td>
<td>Optional Float64</td>
<td>NASDAQ VWOP Price</td>
</tr>
<tr>
<td>BLP_I_SPRD_LAST_RT</td>
<td>Optional Float64</td>
<td>Bloomberg Last I Spread (Real-time)</td>
</tr>
<tr>
<td>PREV_CLOSE_VALUE_REALTIME</td>
<td>Optional Float64</td>
<td>Previous Closing Value</td>
</tr>
<tr>
<td>BID_ALL_SESSION</td>
<td>Optional Float64</td>
<td>Bid Price All Session</td>
</tr>
<tr>
<td>ASK_ALL_SESSION</td>
<td>Optional Float64</td>
<td>Ask Price All Session</td>
</tr>
<tr>
<td>EBS_TOUCH_HIGH_REALTIME</td>
<td>Optional Float64</td>
<td>EBS Touch High</td>
</tr>
<tr>
<td>EBS_TOUCH_LOW_REALTIME</td>
<td>Optional Float64</td>
<td>EBS Touch Low</td>
</tr>
<tr>
<td>PX_PREV_TO_LAST_REALTIME</td>
<td>Optional Float64</td>
<td>Previous-To-Last Price</td>
</tr>
<tr>
<td>PX_TARGIN_SERVICE_REALTIME</td>
<td>Optional Float64</td>
<td>TARGIN Service Price (Real-time)</td>
</tr>
<tr>
<td>PX_TARGIN_OFFICIAL_REALTIME</td>
<td>Optional Float64</td>
<td>TARGIN Official Price (Real-time)</td>
</tr>
<tr>
<td>FOREIGN_HOLDING_PCT_RT</td>
<td>Optional Float64</td>
<td>Percentage Of Foreign Holding (Real-time)</td>
</tr>
<tr>
<td>OWNERSHIP_LIMIT_RATIO_RT</td>
<td>Optional Float64</td>
<td>Ownership Limit Ratio (Real-time)</td>
</tr>
<tr>
<td>RT_EVAL_JAPANESE_CHG_ON_DAY</td>
<td>Optional Float64</td>
<td>Japanese Evaluation Price Change On Day (Real-time)</td>
</tr>
<tr>
<td>RT_EVAL_JAPANESE_PCT_CHG_ON_DAY</td>
<td>Optional Float64</td>
<td>Japanese Eval Price Pct Change On Day (Real-time)</td>
</tr>
<tr>
<td>BLP_Z_SPRD_LAST_RT</td>
<td>Optional Float64</td>
<td>Bloomberg Last Z Spread (Real-time)</td>
</tr>
<tr>
<td>BLP_ASW_SPREAD_LAST_RT</td>
<td>Optional Float64</td>
<td>Bloomberg Last ASW Spread (Real-time)</td>
</tr>
<tr>
<td>BLP_RT_SPRD_TO_BENCH_LAST_RT</td>
<td>Optional Float64</td>
<td>Bloomberg Last Spread to Benchmark (Real-time)</td>
</tr>
<tr>
<td>TRUST_ASK_ORD_VALUE_RT</td>
<td>Optional Float64</td>
<td>Trust Ask Orders Value (Real-time)</td>
</tr>
<tr>
<td>PROPRIETARY_ASK_ORD_VALUE_RT</td>
<td>Optional Float64</td>
<td>Proprietary Ask Orders Value (Real-time)</td>
</tr>
<tr>
<td>TRUST_BID_ORD_VALUE_RT</td>
<td>Optional Float64</td>
<td>Trust Bid Orders Value (Real-time)</td>
</tr>
<tr>
<td>PROPRIETARY_BID_ORD_VALUE_RT</td>
<td>Optional Float64</td>
<td>Proprietary Bid Orders Value (Real-time)</td>
</tr>
<tr>
<td>TOTAL_VALUE_PROGRAM_TRADE_RT</td>
<td>Optional Float64</td>
<td>Total Value of Program Trading (Real-time)</td>
</tr>
<tr>
<td>PX_OFFICIAL_AUCTION_RT</td>
<td>Optional Float64</td>
<td>Official Auction Price (Real-time)</td>
</tr>
<tr>
<td>NYSE_LRP_HIGH_PRICE_RT</td>
<td>Optional Float64</td>
<td>NYSE LRP High Price (Real-time)</td>
</tr>
<tr>
<td>NYSE_LRP_LOW_PRICE_RT</td>
<td>Optional Float64</td>
<td>NYSE LRP Low Price (Real-time)</td>
</tr>
<tr>
<td>ALL_PRICE</td>
<td>Optional Float64</td>
<td>All Price</td>
</tr>
<tr>
<td>BEST_BID1</td>
<td>Optional Float64</td>
<td>Best Bid 1</td>
</tr>
<tr>
<td>BEST_BID2</td>
<td>Optional Float64</td>
<td>Best Bid 2</td>
</tr>
<tr>
<td>BEST_BID3</td>
<td>Optional Float64</td>
<td>Best Bid 3</td>
</tr>
<tr>
<td>BEST_BID4</td>
<td>Optional Float64</td>
<td>Best Bid 4</td>
</tr>
<tr>
<td>BEST_BID5</td>
<td>Optional Float64</td>
<td>Best Bid 5</td>
</tr>
<tr>
<td>BEST_ASK1</td>
<td>Optional Float64</td>
<td>Best Ask 1</td>
</tr>
<tr>
<td>BEST_ASK2</td>
<td>Optional Float64</td>
<td>Best Ask 2</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>BEST_ASK3</td>
<td>Optional Float64</td>
<td>Best Ask 3</td>
</tr>
<tr>
<td>BEST_ASK4</td>
<td>Optional Float64</td>
<td>Best Ask 4</td>
</tr>
<tr>
<td>BEST_ASK5</td>
<td>Optional Float64</td>
<td>Best Ask 5</td>
</tr>
<tr>
<td>BEST_BID1_SZ</td>
<td>Optional Int32</td>
<td>Best Bid 1 Size</td>
</tr>
<tr>
<td>BEST_BID2_SZ</td>
<td>Optional Int32</td>
<td>Best Bid 2 Size</td>
</tr>
<tr>
<td>BEST_BID3_SZ</td>
<td>Optional Int32</td>
<td>Best Bid 3 Size</td>
</tr>
<tr>
<td>BEST_BID4_SZ</td>
<td>Optional Int32</td>
<td>Best Bid 4 Size</td>
</tr>
<tr>
<td>BEST_BID5_SZ</td>
<td>Optional Int32</td>
<td>Best Bid 5 Size</td>
</tr>
<tr>
<td>BEST_ASK1_SZ</td>
<td>Optional Int32</td>
<td>Best Ask 1 Size</td>
</tr>
<tr>
<td>BEST_ASK2_SZ</td>
<td>Optional Int32</td>
<td>Best Ask 2 Size</td>
</tr>
<tr>
<td>BEST_ASK3_SZ</td>
<td>Optional Int32</td>
<td>Best Ask 3 Size</td>
</tr>
<tr>
<td>BEST_ASK4_SZ</td>
<td>Optional Int32</td>
<td>Best Ask 4 Size</td>
</tr>
<tr>
<td>BEST_ASK5_SZ</td>
<td>Optional Int32</td>
<td>Best Ask 5 Size</td>
</tr>
<tr>
<td>LAST_AT_TRADE</td>
<td>Optional Float64</td>
<td>Last AT Trade</td>
</tr>
<tr>
<td>LAST2_AT_TRADE</td>
<td>Optional Float64</td>
<td>Last 2 AT Trade</td>
</tr>
<tr>
<td>LAST_AT_TRADE_TDY</td>
<td>Optional Float64</td>
<td>Today's Last AT Trade</td>
</tr>
<tr>
<td>MID_TDY</td>
<td>Optional Float64</td>
<td>Today's Mid Price</td>
</tr>
<tr>
<td>MID2</td>
<td>Optional Float64</td>
<td>Mid 2 Price</td>
</tr>
<tr>
<td>RT_PX_CHG_NET_1D</td>
<td>Optional Float64</td>
<td>Real-Time Price Change 1 Day Net</td>
</tr>
<tr>
<td>OPEN_TDY</td>
<td>Optional Float64</td>
<td>Today's Open Price</td>
</tr>
<tr>
<td>LAST_PRICE_TDY</td>
<td>Optional Float64</td>
<td>Today's Last Price</td>
</tr>
<tr>
<td>BID_TDY</td>
<td>Optional Float64</td>
<td>Today's Bid Price</td>
</tr>
<tr>
<td>ASK_TDY</td>
<td>Optional Float64</td>
<td>Today's Ask Price</td>
</tr>
<tr>
<td>HIGH_TDY</td>
<td>Optional Float64</td>
<td>Today's High Price</td>
</tr>
<tr>
<td>LOW_TDY</td>
<td>Optional Float64</td>
<td>Today's Low Price</td>
</tr>
<tr>
<td>LAST2_PRICE</td>
<td>Optional Float64</td>
<td>Last 2 Price</td>
</tr>
<tr>
<td>BID2</td>
<td>Optional Float64</td>
<td>Bid 2 Price</td>
</tr>
<tr>
<td>ASK2</td>
<td>Optional Float64</td>
<td>Ask 2 Price</td>
</tr>
<tr>
<td>RT_EXCH_MARKET_STATUS</td>
<td>Optional String</td>
<td>Exchange Market Status</td>
</tr>
<tr>
<td>RT_TRADING_PERIOD</td>
<td>Optional String</td>
<td>Trading Period</td>
</tr>
<tr>
<td>BID_BROKER_CODE</td>
<td>Optional String</td>
<td>Bid Broker Code</td>
</tr>
<tr>
<td>ASK_BROKER_CODE</td>
<td>Optional String</td>
<td>Ask Broker Code</td>
</tr>
<tr>
<td>IMBALANCE_INDIC_RT</td>
<td>Optional String</td>
<td>Imbalance Indicator</td>
</tr>
<tr>
<td>BLP_SPREAD_BENCHMARK_NAME_RT</td>
<td>Optional String</td>
<td>Bloomberg Spread Benchmark Name (Real-time)</td>
</tr>
<tr>
<td>BLP_SWAP_CURVE_NAME_RT</td>
<td>Optional String</td>
<td>Bloomberg Swap Curve Name (Real-time)</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>FINANCIAL_STATUS_INDICATOR_RT</td>
<td>Optional String</td>
<td>Financial Status Indicator (Real-time)</td>
</tr>
<tr>
<td>BID_YLD_COND_CODE</td>
<td>Optional String</td>
<td>Bid Yield Condition Code</td>
</tr>
<tr>
<td>YLD_COND_CODE</td>
<td>Optional String</td>
<td>Yield Condition Code</td>
</tr>
<tr>
<td>ASK_YLD_COND_CODE</td>
<td>Optional String</td>
<td>Ask Yield Condition Code</td>
</tr>
<tr>
<td>ALL_PRICE_COND_CODE</td>
<td>Optional String</td>
<td>Ask Condition Codes</td>
</tr>
<tr>
<td>BID_COND_CODE</td>
<td>Optional String</td>
<td>Bid Condition Codes</td>
</tr>
<tr>
<td>ASK_COND_CODE</td>
<td>Optional String</td>
<td>Ask Condition Codes</td>
</tr>
<tr>
<td>RT_SIMP_SEC_STATUS</td>
<td>Optional String</td>
<td>Simplified Security Status</td>
</tr>
<tr>
<td>RT_PRICING_SOURCE</td>
<td>Optional String</td>
<td>Real-Time Pricing Source</td>
</tr>
<tr>
<td>NYSE_LRP_SEND_TIME_RT</td>
<td>Optional Time</td>
<td>NYSE LRP Send Time (Real-time)</td>
</tr>
<tr>
<td>BID_ASK_TIME</td>
<td>Optional Time</td>
<td>Time of Last Bid/Ask Update</td>
</tr>
<tr>
<td>SES_START</td>
<td>Optional Time</td>
<td>Session Start</td>
</tr>
<tr>
<td>SES_END</td>
<td>Optional Time</td>
<td>Session End</td>
</tr>
<tr>
<td>TRADE_SPREAD_TIME</td>
<td>Optional Time</td>
<td>Time of TRADE_SPREAD tick</td>
</tr>
<tr>
<td>NEWS STORY_TIME</td>
<td>Optional Time</td>
<td>Time of NEWS STORY tick</td>
</tr>
<tr>
<td>BID_TIME</td>
<td>Optional Time</td>
<td>Time of BID tick</td>
</tr>
<tr>
<td>BID_BEST_TIME</td>
<td>Optional Time</td>
<td>Time of BID_BEST tick</td>
</tr>
<tr>
<td>VOLUME_UPDATE_TIME</td>
<td>Optional Time</td>
<td>Time of VOLUME_UPDATE tick</td>
</tr>
<tr>
<td>MARKET_DEPTH_TIME</td>
<td>Optional Time</td>
<td>Time of MARKET_DEPTH tick</td>
</tr>
<tr>
<td>CANCEL_CORRECT_TIME</td>
<td>Optional Time</td>
<td>Time of CANCEL_CORRECT tick</td>
</tr>
<tr>
<td>MIN_LIMIT_OUT_OF_SESSION_TIME</td>
<td>Optional Time</td>
<td>Time of MIN_LIMIT_OUT_OF_SESSION_TIME</td>
</tr>
<tr>
<td>BID_SPREAD_TIME</td>
<td>Optional Time</td>
<td>Time of BID_SPREAD tick</td>
</tr>
<tr>
<td>BT_MKT_TURN_TIME</td>
<td>Optional Time</td>
<td>Time of BT_MKT_TURN tick</td>
</tr>
<tr>
<td>HIGH_TIME</td>
<td>Optional Time</td>
<td>Time of HIGH tick</td>
</tr>
<tr>
<td>BT_LSE_LAST_TIME</td>
<td>Optional Time</td>
<td>Time of BT_LSE_LAST tick</td>
</tr>
<tr>
<td>AT_TRADE_TIME</td>
<td>Optional Time</td>
<td>Time of AT_TRADE tick</td>
</tr>
<tr>
<td>ASK_YEILD_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_YEILD tick</td>
</tr>
<tr>
<td>PRICE_UPDATE_TIME</td>
<td>Optional Time</td>
<td>Time of PRICE_UPDATE tick</td>
</tr>
<tr>
<td>OPEN_INTEREST_TIME</td>
<td>Optional Time</td>
<td>Time of OPEN_INTEREST tick</td>
</tr>
<tr>
<td>VOLUME_TIME</td>
<td>Optional Time</td>
<td>Time of VOLUME tick</td>
</tr>
<tr>
<td>EVAL_JAPANESE_TIME</td>
<td>Optional Time</td>
<td>Time of EVAL_JAPANESE tick</td>
</tr>
<tr>
<td>ASK_WEIGHTED_AVG_SPREAD_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_WEIGHTED_AVG_SPREAD_TIME</td>
</tr>
<tr>
<td>THEO_PRICE_TIME</td>
<td>Optional Time</td>
<td>Time of THEO_PRICE tick</td>
</tr>
<tr>
<td>BUY_SELL_INFO_TIME</td>
<td>Optional Time</td>
<td>Time of BUY_SELL_INFO tick</td>
</tr>
<tr>
<td>SETS_MID_PRICE_TIME</td>
<td>Optional Time</td>
<td>Time of SETS_MID_PRICE tick</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>TAKE_TIME</td>
<td>Optional Time</td>
<td>Time of TAKE tick</td>
</tr>
<tr>
<td>TICK_NUM_TIME</td>
<td>Optional Time</td>
<td>Time of TICK_NUM tick</td>
</tr>
<tr>
<td>SMART_TIME</td>
<td>Optional Time</td>
<td>Time of SMART tick</td>
</tr>
<tr>
<td>INDICATIVE_ASK_TIME</td>
<td>Optional Time</td>
<td>Time of INDICATIVE_ASK tick</td>
</tr>
<tr>
<td>BT_SEC_ASK_TIME</td>
<td>Optional Time</td>
<td>Time of BT_SEC_ASK tick</td>
</tr>
<tr>
<td>LOW_TIME</td>
<td>Optional Time</td>
<td>Time of LOW tick</td>
</tr>
<tr>
<td>BT_SEC_BID_TIME</td>
<td>Optional Time</td>
<td>Time of BT_SEC_BID tick</td>
</tr>
<tr>
<td>LOW_YIELD_TIME</td>
<td>Optional Time</td>
<td>Time of LOW_YIELD tick</td>
</tr>
<tr>
<td>MAX_LIMIT_TIME</td>
<td>Optional Time</td>
<td>Time of MAX_LIMIT tick</td>
</tr>
<tr>
<td>TRADING_PERIOD_TIME</td>
<td>Optional Time</td>
<td>Time of TRADING_PERIOD tick</td>
</tr>
<tr>
<td>INDICATIVE_BID_TIME</td>
<td>Optional Time</td>
<td>Time of INDICATIVE_BID tick</td>
</tr>
<tr>
<td>API_INTERNAL_TIME</td>
<td>Optional Time</td>
<td>Time of API_INTERNAL tick</td>
</tr>
<tr>
<td>ASK_LIFT_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_LIFT tick</td>
</tr>
<tr>
<td>NYSE_LIQUIDITY_ASK_TIME</td>
<td>Optional Time</td>
<td>Time of NYSE_LIQUIDITY_ASK tick</td>
</tr>
<tr>
<td>BID_YEILD_TIME</td>
<td>Optional Time</td>
<td>Time of BID_YEILD tick</td>
</tr>
<tr>
<td>ASK_BEST_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_BEST tick</td>
</tr>
<tr>
<td>MKT_INDICATOR_TIME</td>
<td>Optional Time</td>
<td>Time of MKT_INDICATOR tick</td>
</tr>
<tr>
<td>NYSE_LIQUIDITY_BID_TIME</td>
<td>Optional Time</td>
<td>Time of NYSE_LIQUIDITY_BID tick</td>
</tr>
<tr>
<td>SMART_QUOTE_TIME</td>
<td>Optional Time</td>
<td>Time of SMART_QUOTE tick</td>
</tr>
<tr>
<td>NEW_MKT_DAY_TIME</td>
<td>Optional Time</td>
<td>Time of NEW_MKT_DAY tick</td>
</tr>
<tr>
<td>MAN_TRADE_WITH_SIZE_TIME</td>
<td>Optional Time</td>
<td>Time of MAN_TRADE_WITH_SIZE tick</td>
</tr>
<tr>
<td>BT_ASK_RECAP_TIME</td>
<td>Optional Time</td>
<td>Time of BT_ASK_RECAP tick</td>
</tr>
<tr>
<td>BT_MID_PRICE_TIME</td>
<td>Optional Time</td>
<td>Time of BT_MID_PRICE tick</td>
</tr>
<tr>
<td>BID_MKT_MAKER_TIME</td>
<td>Optional Time</td>
<td>Time of BID_MKT_MAKER tick</td>
</tr>
<tr>
<td>SETTLE_TIME</td>
<td>Optional Time</td>
<td>Time of SETTLE tick</td>
</tr>
<tr>
<td>HIT_TIME</td>
<td>Optional Time</td>
<td>Time of HIT tick</td>
</tr>
<tr>
<td>BT_LAST_RECAP_TIME</td>
<td>Optional Time</td>
<td>Time of BT_LAST_RECAP tick</td>
</tr>
<tr>
<td>LAST_TRADE_TIME</td>
<td>Optional Time</td>
<td>Time of LAST_TRADE</td>
</tr>
<tr>
<td>PRE_POST_MARKET_TIME</td>
<td>Optional Time</td>
<td>Time of PRE_POST_MARKET tick</td>
</tr>
<tr>
<td>ALL_PRICE_TIME</td>
<td>Optional Time</td>
<td>Time of ALL_PRICE tick</td>
</tr>
<tr>
<td>OPEN_TIME</td>
<td>Optional Time</td>
<td>Time of OPEN tick</td>
</tr>
<tr>
<td>HIGH_YEILD_TIME</td>
<td>Optional Time</td>
<td>Time of HIGH_YEILD tick</td>
</tr>
<tr>
<td>ASK_MKT_MAKER_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_MKT_MAKER tick</td>
</tr>
<tr>
<td>MAX_LIMIT_OUT_OF_SESSION_TIME</td>
<td>Optional Time</td>
<td>Time of MAX_LIMIT_OUT_OF_SESSION_tick</td>
</tr>
<tr>
<td>SMARTMAX_TIME</td>
<td>Optional Time</td>
<td>Time of SMARTMAX tick</td>
</tr>
<tr>
<td>YIELD_TIME</td>
<td>Optional Time</td>
<td>Time of YIELD tick</td>
</tr>
<tr>
<td>Element</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>VWAP_TIME</td>
<td>Optional Time</td>
<td>Time of VWAP tick</td>
</tr>
<tr>
<td>BID_WEIGHTED_AVG_SPREAD_TIME</td>
<td>Optional Time</td>
<td>Time of BID_WEIGHTED_AVG_SPREAD tick</td>
</tr>
<tr>
<td>ASK_TIME</td>
<td>Optional Time</td>
<td>Time of ASK tick</td>
</tr>
<tr>
<td>MIN_LIMIT_TIME</td>
<td>Optional Time</td>
<td>Time of MIN_LIMIT tick</td>
</tr>
<tr>
<td>ASK_SPREAD_TIME</td>
<td>Optional Time</td>
<td>Time of ASK_SPREAD tick</td>
</tr>
<tr>
<td>SETTLE_YIELD_TIME</td>
<td>Optional Time</td>
<td>Time of SETTLE_YIELD tick</td>
</tr>
<tr>
<td>BID_LIFT_TIME</td>
<td>Optional Time</td>
<td>Time of BID_LIFT tick</td>
</tr>
<tr>
<td>BT_BID_RECAP_TIME</td>
<td>Optional Time</td>
<td>Time of BT_BID_RECAP tick</td>
</tr>
</tbody>
</table>
### A.5.4 Market VWAP Service Subscription Options

<table>
<thead>
<tr>
<th>Argument Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VWAP_START_TIME</td>
<td>string</td>
<td>Start trade time in the format, HH:MM. HH is in 24-hr format. Only trades at this or past this time are considered for VWAP computation. Specified in TZDF&lt;GO&gt; timing for Desktop API and UTC for Server API.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_START_TIME=11:00&quot;, new CorrelationID(security) );</td>
</tr>
<tr>
<td>VWAP_END_TIME</td>
<td>string</td>
<td>End trade time in the format, HH:MM. HH is in 24-hr format. Only trades at this or before this time are considered for VWAP computation. Specified in TZDF&lt;GO&gt; timing for Desktop API and UTC for Server API.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_END_TIME=12:00&quot;, new CorrelationID(security) ) ;</td>
</tr>
<tr>
<td>VWAP_MIN_SIZE</td>
<td>string</td>
<td>Minimum trade volume for a trade to be included in VWAP computation. Values are taken as signed integers.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_MIN_SIZE=1000&quot;, new CorrelationID(security) ) ;</td>
</tr>
<tr>
<td>VWAP_MAX_SIZE</td>
<td>string</td>
<td>Maximum trade volume for a trade to be included in VWAP computation. Values are taken as signed integers.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_MAX_SIZE=2000&quot;, new CorrelationID(security) ) ;</td>
</tr>
<tr>
<td>VWAP_MIN_PX</td>
<td>string</td>
<td>Minimum trade price for a trade to be included in VWAP computation. Values are taken as floats.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_MIN_PX=23.5&quot;, new CorrelationID(security) ) ;</td>
</tr>
<tr>
<td>VWAP_MAX_PX</td>
<td>string</td>
<td>Maximum trade price for a trade to be included in VWAP computation. Values are taken as floats.</td>
</tr>
<tr>
<td><strong>Example Syntax:</strong></td>
<td></td>
<td>Subscription mySubscription = new Subscription( topic + security, fields, &quot;&amp;VWAP_MAX_PX=25.5&quot;, new CorrelationID(security) ) ;</td>
</tr>
</tbody>
</table>
A.6 Schema for API Authorization

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorizationRequest</td>
<td>Requests Bloomberg to check if a given Bloomberg Anywhere user is logged into the BLOOMBERG PROFESSIONAL service at a specified location.</td>
</tr>
<tr>
<td>UserAsidEquivalenceRequest</td>
<td>Deprecated. Compares the exchanges entitlements of a given user to the exchange entitlements of the ServerAPI.</td>
</tr>
<tr>
<td>LogonStatusRequest</td>
<td>Requests a user's logon status for their Bloomberg Anywhere.</td>
</tr>
<tr>
<td>UserEntitlementsRequest</td>
<td>Requests a list of the user's exchange entitlements</td>
</tr>
<tr>
<td>SecurityEntitlementsRequest</td>
<td>Requests a list of a specific security's exchange entitlements</td>
</tr>
<tr>
<td>SecurityEntitlementsByUserRequest</td>
<td>Deprecated. Requests a list of exchange entitlements for a security by user.</td>
</tr>
<tr>
<td>TokenRequest</td>
<td>Deprecated. Requests a token.</td>
</tr>
</tbody>
</table>

### A.6.1 Authorization Request

**Bloomberg UUID:** the Bloomberg unique user identifier

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td></td>
<td>integer</td>
<td>The Bloomberg unique user identifier</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```java
Request request = authSvc.CreateAuthorizationRequest();
request.Set("uuid", 11223344);
```

**IP Address:** Location of where the user is viewing the ServerAPI data

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipAddress</td>
<td></td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Example Syntax:**

```java
Request authRequest = d_apiAuthSvc.CreateAuthorizationRequest();
authRequest.Set("ipAddress", "111.22.33.44");
```

**Require ASID equivalence:** Deprecated. Sets a flag to check the user has a superset of entitlements compared to the ServerAPI. Used for the All-or-None model of permissioning.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requireAsidEquivalence</td>
<td>TRUE or FALSE</td>
<td>Boolean</td>
<td>When set to 'true', the AuthorizationRequest will succeed only if the users permission are equal to or greater than that of the Server API.</td>
</tr>
</tbody>
</table>

**Example Syntax:**

```java
request.Set("requireAsidEquivalence", true);
```

**Token:** Deprecated. Authorizes the user with the token based approach.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td></td>
<td>Token</td>
<td>Token returned by TokenRequest for a user. (Optional. Either ipAddress or token must be supplied.)</td>
</tr>
</tbody>
</table>
A.6.2 Authorization Request Response

A.6.3 Logon Status Request

**Bloomberg UUID:** the Bloomberg unique user identifier

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td></td>
<td>integer</td>
<td>The Bloomberg Unique User Identifier (UUID)</td>
</tr>
<tr>
<td>sid</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>sidInstance</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>terminalSid</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>terminalSidInstance</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```
Request request = authSvc.CreateRequest("LogonStatusRequest");
Element userinfo = request.GetElement("userInfo");
userinfo.SetElement("uuid", 11223344);
```

**IP Address:** The location where the user is viewing API data

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipAddress</td>
<td></td>
<td>string</td>
<td>The location where the user is viewing API data</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```
Request logonStatusRequest = authSvc.CreateRequest("LogonStatusRequest");
logonStatusRequest.Set("ipAddress", "111.22.33.44");
```

A.6.4 Logon Status Request Response

A.6.5 User Entitlements Request

Bloomberg UUID: the Bloomberg unique user identifier

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>integer</td>
<td></td>
<td>The Bloomberg Unique User Identifier (UUID)</td>
</tr>
<tr>
<td>sid</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>sidInstance</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>terminalSid</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
<tr>
<td>terminalSidInstance</td>
<td></td>
<td></td>
<td>Deprecated. do not use</td>
</tr>
</tbody>
</table>

Example Syntax:

```java
Request request = authSvc.CreateRequest("UserEntitlementsRequest");
Element userinfo = request.GetElement("userInfo");
userinfo.SetElement("uuid", 11223344);
```

A.6.6 User Entitlements Request Response

A.6.7 Security Entitlements Request

<table>
<thead>
<tr>
<th>Securities: the reference or streaming fields desired.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>securities</td>
</tr>
</tbody>
</table>

**Example Syntax:**
```java
Request request = authSvc.CreateRequest("SecurityEntitlementsRequest");
Element securities = request.GetElement("securities");
securities.AppendValue("IBM US Equity");
```

A.6.8 Security Entitlements Request Response

A.6.9 Authorization Token Request

Identifier: The Bloomberg Unique User Identifier.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td></td>
<td>integer</td>
<td>The Bloomberg Unique User Identifier (UUID)</td>
</tr>
</tbody>
</table>

Example Syntax:
```
Request request = authSvc.CreateRequest("AuthorizationTokenRequest");
request.Set("uuid", 11223344);
```

Label: A label that identifies which Server API application is requesting the token.

<table>
<thead>
<tr>
<th>Element</th>
<th>Element Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td></td>
<td>string</td>
<td>String identifier for the requesting ServerAPI application</td>
</tr>
</tbody>
</table>

Example Syntax:
```
Request request = authSvc.CreateRequest("AuthorizationTokenRequest");
request.Set("label", "myApp");
```

A.6.10 Authorization Token Request Response

### A.6.11 Field Service Response Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorizationSuccess</td>
<td>Returned for an authorization request when the UUID provided is logged into the Bloomberg Anywhere at the specified IP address.</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>Returned for an authorization request on failure. It is an errorInfo element.</td>
</tr>
<tr>
<td>reason</td>
<td>An AuthorizationFailure message will contain one &quot;reason&quot; element</td>
</tr>
<tr>
<td>responseError</td>
<td>Returned when a request cannot be completed for any reason. It is an errorInfo element.</td>
</tr>
<tr>
<td>errorInfo</td>
<td>Contains values about the error which has occurred, including the source, code, category, message, and subcategory.</td>
</tr>
<tr>
<td>eidData[ ]</td>
<td>Contains a list of eidData elements, each associated to a security requested.</td>
</tr>
<tr>
<td>eidData[ ]::eidData</td>
<td>Contains status, sequence number and list of entitlement identifiers.</td>
</tr>
<tr>
<td>eids[ ]</td>
<td>Contains a list of entitlementId values associated to the user.</td>
</tr>
</tbody>
</table>

### A.6.12 Field Service Request Values

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>String</td>
<td>Bloomberg internal error source information.</td>
</tr>
<tr>
<td>Code</td>
<td>Integer</td>
<td>Bloomberg internal error code.</td>
</tr>
<tr>
<td>Category</td>
<td>String</td>
<td>Bloomberg error classification. Used to determine the general classification of the failure.</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Human readable description of the failure.</td>
</tr>
<tr>
<td>subcategory</td>
<td>String</td>
<td>(Optional) Bloomberg sub-error classification. Used to determine the specific classification of the failure.</td>
</tr>
<tr>
<td>entitlementId</td>
<td>Integer</td>
<td>Entitlement identifier (EID)</td>
</tr>
<tr>
<td>status</td>
<td>Integer</td>
<td>Status where success = 0. Any other code indicates failure.</td>
</tr>
<tr>
<td>sequenceNumber</td>
<td>Integer</td>
<td>Security sequence number, specifying the position of the security in the request.</td>
</tr>
<tr>
<td>isLoggedOn</td>
<td>Boolean</td>
<td>Returns true when the UUID specified in logged into the BLOOMBERG PROFESSIONAL service at the specified IP address.</td>
</tr>
</tbody>
</table>
B  Error Codes

See the following sections:
- General
- //BLP/APIAUTH
- //BLP/MKTDATA and //BLP/MKTVWAP
- //BLP/REFDATA
- //BLP/MKTDATA and //BLP/MKTVWAP

B.1 General

Event Name: ADMIN

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving the message indicates client is slow. NO category/subcategory.</td>
<td>SlowConsumerWarning</td>
<td></td>
</tr>
<tr>
<td>Receiving the message indicates client is not slow anymore. NO category/subcategory.</td>
<td>SlowConsumerWarningCleared</td>
<td></td>
</tr>
</tbody>
</table>

Event Name: SESSION_STATUS

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Session has been started successfully</td>
<td>SessionStarted</td>
<td></td>
</tr>
<tr>
<td>The Session has been terminated</td>
<td>SessionTerminated</td>
<td>IO_ERROR</td>
</tr>
<tr>
<td>The Session has been failed to start</td>
<td>SessionStartupFailure</td>
<td>IO_ERROR</td>
</tr>
<tr>
<td>Session is up either because Session.Start() was called or that the connection between the application and the Bloomberg Communication Server process (e.g. ServerApi, B-Pipe) has been re-established.</td>
<td>SessionConnectionUp</td>
<td>IO_ERROR</td>
</tr>
<tr>
<td>Session is down either because Session.Stop() was called or that the connection between the application and the Bloomberg Communication Server process (e.g. ServerApi, B-Pipe) is lost.</td>
<td>SessionConnectionDown</td>
<td>IO_ERROR</td>
</tr>
</tbody>
</table>
## Event Name: SERVICE_STATUS

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Service has been opened successfully</td>
<td>ServiceOpened</td>
<td></td>
</tr>
<tr>
<td>The Service has failed to open (I/O Error)</td>
<td>ServiceOpenFailure</td>
<td>IO_ERROR</td>
</tr>
<tr>
<td>The Service has failed to open (Other)</td>
<td>ServiceOpenFailure</td>
<td>UNCLASSIFIED</td>
</tr>
</tbody>
</table>

### B.2 //BLP/APIAUTH

#### B.2.1 AUTHORIZATION_STATUS, REQUEST_STATUS, RESPONSE and PARTIAL_RESPONSE Events

**Request: AuthorizationRequest**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>User was authorized successfully.</td>
<td>AuthorizationSuccess</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User is not logged into Bloomberg.</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>NOT_LOGGED_IN</td>
</tr>
<tr>
<td>Invalid User ID.</td>
<td>AuthorizationFailure</td>
<td>BAD_ARGS</td>
<td>INVALID_USER</td>
</tr>
<tr>
<td>Valid User ID belonging to different firm.</td>
<td>ResponseError</td>
<td>NO_AUTH</td>
<td>CROSS_FIRM_AUTH</td>
</tr>
<tr>
<td>Invalid Display (when IP is specified)</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>INVALID_DISPLAY</td>
</tr>
<tr>
<td>Timeout waiting for input or expired token.</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>TOKEN_EXPIRED</td>
</tr>
<tr>
<td>Bad unparsable token supplied.</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>BAD_AUTH_TOKEN</td>
</tr>
<tr>
<td>User cancels request (Launchpad).</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>CANCELLED_BY_USER</td>
</tr>
<tr>
<td>UserAsidEquivalence check failed</td>
<td>AuthorizationFailure</td>
<td>NO_AUTH</td>
<td>ENTITLEMENTS_MISMATCH</td>
</tr>
<tr>
<td>No token and IP specified</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>N/A</td>
</tr>
<tr>
<td>User has logged off and then back onto the Bloomberg Professional service. The user's Identity object remains valid. message = &quot;User re-logged on&quot;</td>
<td>EntitlementChanged</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
**Scenario** | **Message Type** | **Category** | **Sub-Category**
---|---|---|---
The entitlements of the User/Application have been changed in EMRS. Will usually take an hour to take effect and, therefore, to generate the message. The user/application’s Identity object remains valid. | EntitlementChanged | N/A | N/A
A user logs into a Bloomberg Professional service other than the one on the PC he is running his application. | AuthorizationRevoked | NO_AUTH | INVALID_DISPLAY
When user uses an API that is either deprecated or passes parameters in an authorization request that are not supported for the specific product in question. For example, emrsname + IP authorization is not supported for ServerApi. Similarly UUID+IP authorizations are not supported on platforms. A descriptive error message is returned in the latter case. | AuthorizationFailure | NOT_AVAILABLE | NOT_AVAILABLE_API
User locked out of the Bloomberg Professional service. | AuthorizationRevoked | NO_AUTH | LOCKOUT
This is sent when deactivating the application in EMRS after it had been used to authenticate in APPLICATION_ONLY mode. It is also sent when unchecking the activate checkbox in EMRS for the user after it had been authenticated. | AuthorizationRevoked | NO_AUTH | CANCELED_BY_SERVER
A user logs into a Bloomberg Professional service other than the one on the PC he is running his application. | AuthorizationRevoked | NO_AUTH | INVALID_DISPLAY
### B.2.2 REQUEST_STATUS, RESPONSE and PARTIAL_RESPONSE Events (B-Pipe ONLY)

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Scenario</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorizationUpdate</td>
<td>User logged in to another Bloomberg Professional service.</td>
<td>NO_AUTH</td>
<td>INVALID_DISPLAY</td>
</tr>
<tr>
<td>AuthorizationUpdate</td>
<td>User Locked out of Bloomberg Professional service. Click here for further details.</td>
<td>NO_AUTH</td>
<td>LOCKOUT</td>
</tr>
<tr>
<td>AuthorizationUpdate</td>
<td>The authorization was cancelled by the server through EMRS administrator.</td>
<td>UNCLASSIFIED</td>
<td>CANCELLED_BY_SERVER</td>
</tr>
<tr>
<td>AuthorizationRequest</td>
<td>The user is not permitted to use the application.</td>
<td>NO_AUTH</td>
<td>NO_APP_PERM</td>
</tr>
<tr>
<td>AuthorizationRequest</td>
<td>The requested authorization type is not supported for this ASID type.</td>
<td>NO_AUTH</td>
<td>INVALID_ASID_TYPE</td>
</tr>
<tr>
<td>AuthorizationRequest</td>
<td>Your authorization token has been used by another instance.</td>
<td>NO_AUTH</td>
<td>CREDENTIAL_REUSE</td>
</tr>
<tr>
<td>AuthorizationRequest</td>
<td>The token has expired. You must regenerate the token and authorize.</td>
<td>NO_AUTH</td>
<td>EXPIRED_AUTHTOKEN</td>
</tr>
<tr>
<td>AuthorizationRequest</td>
<td>The maximum number of devices for this seat type has been exceeded.</td>
<td>LIMIT</td>
<td>MAX_DEVICES_EXCEEDED</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>Exceeded maximum number of simultaneous authorizations.</td>
<td>LIMIT</td>
<td>n/a</td>
</tr>
<tr>
<td>AuthorizationUpdate</td>
<td>Entity/ASID delivery point not enabled in EMRS. You will receive this error if a failure is dynamically detected because someone changed EMRS, and an existing authorization is affected, after the authorization had been successfully made.</td>
<td>NO_AUTH</td>
<td>EMRS_ENTITY_ASID_MISMATCH</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>Entity/ASID combination not enabled in EMRS. You will receive this error if this failure is detected at authorization time.</td>
<td>NO_AUTH</td>
<td>EMRS_ENTITY_ASID_MISMATCH</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>Application IP mismatch with EMRS IP ranges.</td>
<td>NO_AUTH</td>
<td>EMRS_IPRANGE_MISMATCH</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>User or Application is not enabled for datafeed (B-Pipe) access in EMRS and attempting to authorize using a B-Pipe.</td>
<td>NO_AUTH</td>
<td>EMRS_DATAFEED_DISABLED</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>User or Application is not enabled for platform access in EMRS and attempting to authorize using a DDM.</td>
<td>NO_AUTH</td>
<td>EMRS_PLATFORM_DISABLED</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>The application has no instance created for the B-Pipe instance (delivery point) in EMRS.</td>
<td>NO_AUTH</td>
<td>INVALID_DELIVERY_POINT</td>
</tr>
<tr>
<td>AuthorizationFailure</td>
<td>The Application is authorizing from a machine whose IP is being prevented by the IP Restrictions configured in EMRS.</td>
<td>NO_AUTH</td>
<td>IP_NOT_IN_RANGE</td>
</tr>
</tbody>
</table>
B.2.3 TOKEN_STATUS Event (B-Pipe ONLY)

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Scenario</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenGenerationSuccess</td>
<td>A token was successfully generated.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TokenGenerationFailure</td>
<td>Library or backend errors</td>
<td>NO_AUTH</td>
<td>INTERNAL_ERROR</td>
</tr>
<tr>
<td>TokenGenerationFailure</td>
<td>The user cannot be found in the EMRS database.</td>
<td>NO_AUTH</td>
<td>INVALID_USER</td>
</tr>
<tr>
<td>TokenGenerationFailure</td>
<td>The application name cannot be found in the EMRS database.</td>
<td>NO_AUTH</td>
<td>INVALID_APP</td>
</tr>
<tr>
<td>TokenGenerationFailure</td>
<td>The firm number mismatches with user(s) or application(s).</td>
<td>NO_AUTH</td>
<td>CROSS_FIRM_AUTH</td>
</tr>
<tr>
<td>TokenGenerationSuccess</td>
<td>A token was successfully generated.</td>
<td>BAD_ARGS</td>
<td>INVALID_USER or INVALID_APP</td>
</tr>
<tr>
<td>TokenGenerationFailure</td>
<td>A token was not successfully generated.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B.3 //BLP/MKTDATA and //BLP/MKT VWAP

B.3.1 (SUBSCRIPTION_DATA and SUBSCRIPTION_STATUS Events)

Request: session.subscribe

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent subscription limit has been exceeded.</td>
<td>SubscriptionTerminated</td>
<td>LIMIT</td>
</tr>
<tr>
<td>&quot;Failed to obtain initial paint&quot;</td>
<td>SubscriptionTerminated</td>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>If this error occurs, the Bloomberg Data Center was unable to get the initial paint for the subscription. You will still receive subscription ticks.</td>
<td>SubscriptionTerminated</td>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>Subscription has been canceled via Unsubscribe() or Cancel() call.</td>
<td>SubscriptionTerminated</td>
<td>CANCELED</td>
</tr>
<tr>
<td>Subscription has been started.</td>
<td>SubscriptionStarted</td>
<td></td>
</tr>
<tr>
<td>Invalid user or credentials or user is being blocked by metering server.</td>
<td>SubscriptionFailure</td>
<td>NO_AUTH</td>
</tr>
<tr>
<td>Bad Topic string or Service name in Topic</td>
<td>SubscriptionFailure</td>
<td>BAD_TOPIC</td>
</tr>
<tr>
<td>Bad Security</td>
<td>SubscriptionFailure</td>
<td>BAD_SEC</td>
</tr>
<tr>
<td>Not a real-time security (no streamId or monid).</td>
<td>SubscriptionFailure</td>
<td>NOT_MONITORABLE</td>
</tr>
<tr>
<td>Field not valid to the specified security.</td>
<td>SubscriptionFailure</td>
<td>NOT_APPLICABLE</td>
</tr>
<tr>
<td>Invalid field</td>
<td>SubscriptionFailure</td>
<td>BAD_FLD</td>
</tr>
<tr>
<td>Request timed-out.</td>
<td>SubscriptionFailure</td>
<td>TIMEOUT</td>
</tr>
<tr>
<td>Invalid field. No permissions.</td>
<td>SubscriptionFailure</td>
<td>UNCLASSIFIED</td>
</tr>
</tbody>
</table>
**B.4 //BLP/REFDATA**

### B.4.1 For All Requests

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daily limit for user reached.</td>
<td>ResponseError</td>
<td>LIMIT</td>
<td>DAILY_LIMIT_REACHED</td>
</tr>
<tr>
<td>Monthly limit for user reached.</td>
<td>ResponseError</td>
<td>LIMIT</td>
<td>MONTHLY_LIMIT_REACHED</td>
</tr>
<tr>
<td>Manually disabled user</td>
<td>ResponseError</td>
<td>LIMIT</td>
<td>MANUALLY_DISABLED</td>
</tr>
<tr>
<td>FTT limit reached</td>
<td>ResponseError</td>
<td>LIMIT</td>
<td>FREE_TRIAL_TERM_LIMIT_REACHED</td>
</tr>
<tr>
<td>Invalid ASID or User</td>
<td>ResponseError</td>
<td>NO_AUTH</td>
<td>INVALID_USER</td>
</tr>
<tr>
<td>No products found (SAPI only)</td>
<td>ResponseError</td>
<td>NO_AUTH</td>
<td>NO_PRODUCTS_FOUND</td>
</tr>
<tr>
<td>User logged in from different firm.</td>
<td>ResponseError</td>
<td>NO_AUTH</td>
<td>CROSS_FIRM_AUTH</td>
</tr>
<tr>
<td>Invalid security requested</td>
<td>ResponseError</td>
<td>BAD_SEC</td>
<td>INVALID_SECURITY_IDENTIFIER</td>
</tr>
<tr>
<td>Internal error</td>
<td>ResponseError</td>
<td>UNCLASSIFIED</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

### B.4.2 HistoricalDataRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid start/end date requested.</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>INVALID_START_END</td>
</tr>
<tr>
<td>Invalid currency requested.</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>INVALID_CURRENCY</td>
</tr>
<tr>
<td>No Fields requested.</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>NO_FIELDS</td>
</tr>
<tr>
<td>Requested too many Fields.</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>TOO_MANY_FIELDS</td>
</tr>
<tr>
<td>Invalid Field</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>INVALID_FIELD</td>
</tr>
<tr>
<td>Invalid override Field requested.</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>INVALID_OVERRIDE_FIELD</td>
</tr>
<tr>
<td>Not valid historical Field requested.</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>NOT_APPLICABLE_TO_HIST_DATA</td>
</tr>
<tr>
<td>Historical Field is not applicable to market sector.</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>NOT_APPLICABLE_TO_SECTOR</td>
</tr>
<tr>
<td>No data currently available.</td>
<td>ResponseError</td>
<td>NOTAVAILABLE</td>
<td>NOT_AVAILABLE_API</td>
</tr>
</tbody>
</table>
### B.4.3 IntradayBarRequest and IntradayTickRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>No event type requested.</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>NO_EVENT_TYPE</td>
</tr>
</tbody>
</table>

### B.4.4 ReferenceDataRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
<th>Sub-Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid field</td>
<td>ResponseError</td>
<td>NOT_AVAILABLE</td>
<td>INVALID_FIELD_DATA</td>
</tr>
<tr>
<td>Too many override fields requested</td>
<td>ResponseError</td>
<td>BAD_ARGS</td>
<td>TOO_MANY_OVERRIDES</td>
</tr>
<tr>
<td>Invalid override field requested</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>INVALID_OVERRIDE_FIELD</td>
</tr>
<tr>
<td>Not valid refdata field requested</td>
<td>ResponseError</td>
<td>BAD_FLD</td>
<td>NOT_APPLICABLE_TO_REF_DATA</td>
</tr>
</tbody>
</table>

### B.4.5 categorizedFieldSearchRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Bloomberg.</td>
<td>categorizedFieldResponse</td>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td></td>
<td>categorizedFieldSearchError</td>
<td>BAD_ARGS</td>
</tr>
<tr>
<td>Invalid request/no search string</td>
<td>categorizedFieldResponse</td>
<td>BAD_FLD</td>
</tr>
<tr>
<td></td>
<td>categorizedFieldSearchError</td>
<td></td>
</tr>
</tbody>
</table>

### B.4.6 fieldInfoRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact Bloomberg.</td>
<td>fieldResponse</td>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>Some field IDs are invalid.</td>
<td>fieldResponse.fieldData.field</td>
<td></td>
</tr>
</tbody>
</table>

### B.4.7 fieldSearchRequest

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Message Type</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid request/Invalid Field IDs</td>
<td>fieldSearchError</td>
<td></td>
</tr>
<tr>
<td></td>
<td>fieldResponse.fieldSearchError</td>
<td></td>
</tr>
</tbody>
</table>
This section contains the following code examples and sample output from each example:

- “Request Response Paradigm” on page 228
- “Subscription Paradigm” on page 231
- “Asynchronous Event Handling” on page 235
- “Request Response Multiple” on page 239
- “Subscription Multiple” on page 243
- “Authorization by IP Address” on page 253
C.1 Request Response Paradigm

```java
// RequestResponseParadigm.java

package BloombergLP;

import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Request;
import com.bloomberglp.blpapi.Service;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;

public class RequestResponseParadigm {

    public static void main(String[] args) throws Exception {
        SessionOptions sessionOptions = new SessionOptions();
        sessionOptions.setServerHost("localhost");
        sessionOptions.setServerPort(8194);
        Session session = new Session(sessionOptions);
        if (!session.start()) {
            System.out.println("Could not start session.");
            System.exit(1);
        }
        if (!session.openService("//blp/refdata")) {
            System.out.println("Could not open service " +
                                "//blp/refdata");
            System.exit(1);
        }
        CorrelationID requestID = new CorrelationID(1);
        Service refDataSvc = session.getService("//blp/refdata");
        Request request = refDataSvc.createRequest("ReferenceDataRequest");
        request.append("securities", "IBM US Equity");
        request.append("fields", "PX_LAST");
        session.sendRequest(request, requestID);
    }
}
```
boolean continueToLoop = true;
while (continueToLoop) {
    Event event = session.nextEvent();
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.RESPONSE: // final event
        continueToLoop = false;               // fall through
        break;
    case Event.EventType.Constants.PARTIAL_RESPONSE:
        handleResponseEvent(event);
        break;
    default:
        handleOtherEvent(event);
        break;
    }
}

private static void handleResponseEvent(Event event) throws Exception {
    System.out.println("EventType =" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=" +
                             message.correlationID());
        System.out.println("messageType =" +
                             message.messageType());
        message.print(System.out);
    }
}

private static void handleOtherEvent(Event event) throws Exception {
    System.out.println("EventType=" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=" +
                             message.correlationID());
        System.out.println("messageType=" + message.messageType());
        message.print(System.out);
        if (Event.EventType.Constants.SESSION_STATUS ==
            event.eventType().intValue() &&
            "SessionTerminated" ==
            message.messageType().toString()){
            System.out.println("Terminating: " +
                                  message.messageType());
            System.exit(1);
        }
    }
}
C.1.1 Request Response Paradigm Output

```
EventType=SESSION_STATUS
  correlationID=null
  messageType=SessionStarted
  SessionStarted = {}

EventType=SERVICE_STATUS
  correlationID=Internal: 1
  messageType=ServiceOpened
  ServiceOpened = {}

EventType=RESPONSE
  correlationID=User: 1
  messageType=ReferenceDataResponse
  ReferenceDataResponse (choice) = {
    securityData[] = {
      securityData = {
        security = IBM US Equity
        sequenceNumber = 0
        fieldData = {
          PX_LAST = 92.51
        }
      }
    }
  }
```
C.2 Subscription Paradigm

```java
// SubscriptionParadigm.java

package BloombergLP;

import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;
import com.bloomberglp.blpapi.Subscription;
import com.bloomberglp.blpapi.SubscriptionList;

public class SubscriptionParadigm {

    public static void main(String[] args) throws Exception {
        SessionOptions sessionOptions = new SessionOptions();
        sessionOptions.setServerHost("localhost");
        sessionOptions.setServerPort(8194);
        Session session = new Session(sessionOptions);
        if (!session.start()) {
            System.out.println("Could not start session.");
            System.exit(1);
        }
        if (!session.openService("//blp/mktdata")) {
            System.err.println("Could not start session.");
            System.exit(1);
        }
        CorrelationID subscriptionID = new CorrelationID(2);
        SubscriptionList subscriptions = new SubscriptionList();
        subscriptions.add(new Subscription("AAPL US Equity",
                                              "LAST_PRICE",
                                              subscriptionID));
        session.subscribe(subscriptions);
    }

}```
int updateCount = 0;
while (true) {
    Event event = session.nextEvent();
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.SUBSCRIPTION_DATA:
        handleDataEvent(event, updateCount);
        break;
    default:
        handleOtherEvent(event);
        break;
    }
}

private static void handleDataEvent(Event event, int updateCount)
throws Exception {
    System.out.println("EventType=" + event.eventType());
    System.out.println("updateCount = " + updateCount);
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID = " +
                           message.correlationID());
        System.out.println("messageType = " +
                           message.messageType());
        message.print(System.out);
    }
}

private static void handleOtherEvent(Event event) throws Exception {
    System.out.println("EventType=" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=" +
                           message.correlationID());
        System.out.println("messageType=" + message.messageType());
        message.print(System.out);
        if (Event.EventType.Constants.SESSION_STATUS ==
           event.eventType().intValue() &&
           "SessionTerminated" ==
           message.messageType().toString()){
            System.out.println("Terminating: " +
                               message.messageType());
            System.exit(1);
        }
    }
}

EventType=SESSION_STATUS
  correlationID=null
  messageType=SessionStarted
  SessionStarted = {
  }
EventType=SERVICE_STATUS
  correlationID=Internal: 1
  messageType=ServiceOpened
  ServiceOpened = {
  }
EventType=SUBSCRIPTION_STATUS
  correlationID=User: 2
  messageType=SubscriptionStarted
  SubscriptionStarted = {
  }
EventType=SUBSCRIPTION_DATA
  updateCount = 0
  correlationID = User: 2
  messageType = MarketDataEvents
  MarketDataEvents = {
    LAST_PRICE = 93.0
    BID = 92.92
    ASK = 92.95
    VOLUME = 21168694
    HIGH = 94.34
    LOW = 92.6
    RT_OPEN_INTEREST = 31212534
    BEST_BID = 92.92
    BEST_ASK = 92.95
    LAST_TRADE = 93.0
    OPEN = 93.09
    PREV_SES_LAST_PRICE = 94.2
    VWAP = 93.3075
    TRADING_DT_REALTIME = 2009-01-29+00:00
    EQY_TURNOVER_REALTIME = 1.98702464E9
    RT_API_MACHINE = n119
    SES_START = 14:30:00.000+00:00
    SES_END = 21:30:00.000+00:00
    RT_PX_CHG_NET_1D = -1.2
    RT_PX_CHG_PCT_1D = -1.27389
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
    OPEN_TDY = 93.09
    ASK_SIZE_TDY = 1
    BID_SIZE_TDY = 1
    VOLUME_TDY = 21168694
LAST_PRICE_TDY = 93.0
BID_TDY = 92.92
ASK_TDY = 92.95
HIGH_TDY = 94.34
LOW_TDY = 92.6
RT_PRICING_SOURCE = US
ASK_SIZE = 1
BID_SIZE = 1
TIME = 22:20:00.000+00:00
API_MACHINE = n119
EXCH_CODE_LAST = D
EXCH_CODE_BID = Q
EXCH_CODE_ASK = O
EID = 14005
IS_DELAYED_STREAM = false
}
EventType=SUBSCRIPTION_DATA
updateCount = 1
correlationID = User: 2
messageType = MarketDataEvents
MarketDataEvents = {
    LAST_ALL_SESSIONS = 93.0
    BID_ALL_SESSION = 92.92
    ASK_ALL_SESSION = 92.95
    TRADE_SIZE_ALL_SESSIONS_RT = 0
    IS_DELAYED_STREAM = false
}
C.3 Asynchronous Event Handling

// AsynchronousEventHandling.java

package BloombergLP;

import java.io.IOException;
import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.EventHandler;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Request;
import com.bloomberglp.blpapi.Service;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;

public class AsynchronousEventHandling {
    public static void main(String[] args) throws Exception {
        SessionOptions options = new SessionOptions();
        options.setServerHost("localhost");
        options.setServerPort(8194);
        Session session = new Session(sessionOptions, new MyEventHandler());
        session.startAsync();
        // Wait for events
        Object object = new Object();
        synchronized (object) {
            object.wait();
        }
    }
}


```java
class MyEventHandler implements EventHandler {

    void dumpEvent(Event event){
        System.out.println("eventType=" + event.eventType());
        MessageIterator messageIterator = event.messageIterator();
        while (messageIterator.hasNext()){
            Message message = messageIterator.next();
            System.out.println("messageType=" + message.messageType());
            System.out.println("CorrelationID=" + message.correlationID());
            try {
                message.print(System.out);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    public void processEvent(Event event, Session session) {
        switch (event.eventType().intValue()) {
            case Event.EventType.Constants.SESSION_STATUS: {
                MessageIterator iter = event.messageIterator();
                while (iter.hasNext()) {
                    Message message = iter.next();
                    if (message.messageType().equals("SessionStarted")) {
                        try {
                            session.openServiceAsync("//blp/refdata",
                                new CorrelationID(99));
                        } catch (Exception e) {
                            System.err.println(
                                "Could not open //blp/refdata for async");
                            System.exit(1);
                        }
                    } else {
                        System.err.println("Could not start session.");
                        System.exit(1);
                    }
                }
            }
            break;
        }
    }
}
```
case Event.EventType.Constants.SERVICE_STATUS: {
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        if (message.correlationID().value() == 99
                && message.messageType().equals("ServiceOpened")) {
            // Construct and issue a Request
            Service service = session.getService("/blp/refdata");
            Request request =
                service.createRequest("ReferenceDataRequest");
            request.append("securities", "IBM US Equity");
            request.append("fields", "LAST_PRICE");
            try {
                session.sendRequest(request, new CorrelationID(86));
            } catch (Exception e) {
                System.err.println("Could not send request");
                System.exit(1);
            }
        } else {
            System.out.println("Unexpected SERVICE_STATUS message:");
            try {
                message.print(System.err);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    break;
}
```java
    case Event.EventType.Constants.PARTIAL_RESPONSE: { //
        dumpEvent(event); // Handle Partial Response
        break;
    }
    case Event.EventType.Constants.RESPONSE:{
        dumpEvent(event); // Handle final response
        // Now, the example is complete. Shut it down.
        try {
            session.stop(Session.StopOption.ASYNC);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("terminate process from handler");
        System.exit(0);
        break;
    }
    default: {
        System.err.println("unexpected Event");
        dumpEvent(event);
        System.exit(1);
        break;
    }
}
```

C.3.1 Asynchronous Event Handling: Output

```
eventType=RESPONSE
messageType=ReferenceDataResponse
CorrelationID=User: 86
ReferenceDataResponse {choice} = {
    securityData[] = {
        securityData = {
            security = IBM US Equity
            sequenceNumber = 0
            fieldData = {
                LAST_PRICE = 92.51
            }
        }
    }
} 
terminate process from handler
```
C.4 Request Response Multiple

```java
// RequestResponseMultiple.java

package BloombergLP;

import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Element;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Request;
import com.bloomberglp.blpapi.Service;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;

public class RequestResponseMultiple {

    public static void main(String[] args) throws Exception {
        SessionOptions sessionOptions = new SessionOptions();
        sessionOptions.setServerHost("localhost");
        sessionOptions.setServerPort(8194);
        Session session = new Session(sessionOptions);
        if (!session.start()) {
            System.out.println("Could not start session.");
            System.exit(1);
        }
        if (!session.openService("//blp/refdata")) {
            System.out.println("Could not open service "+
                                  "//blp/refdata");
            System.exit(1);
        }
        Service refDataSvc = session.getService("//blp/refdata");
        Request request = refDataSvc.createRequest("ReferenceDataRequest");
        request.getElement("securities").appendValue("AAPL US Equity");
        request.getElement("securities").appendValue("IBM US Equity");
        request.getElement("securities").appendValue("BLAHBLAHBLAH US Equity");
        request.getElement("fields").appendValue("PX_LAST"); // Last Price
        request.getElement("fields").appendValue("DS002"); // Description
        request.getElement("fields").appendValue("VWAP_VOLUME");
        // Volume used to calculate the Volume Weighted Average Price
        // (VWAP)
        session.sendRequest(request, new CorrelationID(1));
    }
}
```
boolean continueToLoop = true;
while (continueToLoop) {
    Event event = session.nextEvent();
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.RESPONSE: // final response
        continueToLoop = false; // fall through
        case Event.EventType.Constants.PARTIAL_RESPONSE:
            handleResponseEvent(event);
            break;
    default:
        handleOtherEvent(event);
        break;
    }
}

private static void handleResponseEvent(Event event) throws Exception {
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        Element ReferenceDataResponse = message.asElement();
        if (ReferenceDataResponse.hasElement("responseError")) {
            System.exit(1);
        }
        Element securityDataArray =
            ReferenceDataResponse.getElement("securityData");
        int numItems = securityDataArray.numValues();
        for (int i = 0; i < numItems; ++i) {
            Element securityData = securityDataArray.getValueAsElement(i);
            String security = securityData.getElementAsString("security");
            int sequenceNumber = securityData.getEleme
private static void handleOtherEvent(Event event) throws Exception {
    System.out.println("EventType=" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID=" + message.correlationID());
        System.out.println("messageType=" + message.messageType());
        message.print(System.out);
        if (Event.EventType.Constants.SESSION_STATUS ==
            event.eventType().intValue() &&
            "SessionTerminated" ==
            message.messageType().toString()){
            System.out.println("Terminating: " + message.messageType());
            System.exit(1);
        }
    }
}
C.4.1 Request Response Multiple: Output

```
EventType=SESSION_STATUS
correlationID=null
messageType=SessionStarted
SessionStarted = {
}
EventType=SERVICE_STATUS
correlationID=Internal: 1
messageType=ServiceOpened
ServiceOpened = {
  * security = AAPL US Equity
  * sequenceNumber=0
  * px_last = 93.0
  * ds002 = APPLE INC
  * vwap_volume = 2.0799279E7

  * security = IBM US Equity
  * sequenceNumber=1
  * px_last = 92.51
  * ds002 = INTL BUSINESS MACHINES CORP
  * vwap_volume = 8916238.0

  * security = BLAHLAHBLAH US Equity
securityError = {
    source = 193::bbdbsl
    code = 15
    category = BAD_SEC
    message = Unknown/Invalid security [nid:193]
    subcategory = INVALID_SECURITY
  }
```
// SubscriptionMultiple.java

package BloombergLP;

import java.io.IOException;
import java.io.PrintStream;

import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.EventHandler;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;
import com.bloomberglp.blpapi.Subscription;
import com.bloomberglp.blpapi.SubscriptionList;

class SubscriptionEventHandler implements EventHandler {
    private String d_label;
    private PrintStream d_printStream;

    // CREATORS
    SubscriptionEventHandler(String label, PrintStream printStream) {
        d_label = label;
        d_printStream = printStream;
    }

    // MANIPULATORS
    public void processEvent(Event event, Session session) {
        switch (event.eventType().intValue()) {
        case Event.EventType.Constants.SUBSCRIPTION_DATA:
            handleDataEvent(event, session);
            break;
        case Event.EventType.Constants.SESSION_STATUS:
        case Event.EventType.Constants.SERVICE_STATUS:
        case Event.EventType.Constants.SUBSCRIPTION_STATUS:
            handleStatusEvent(event, session);
            break;
        default:
            handleOtherEvent(event, session);
            break;
        }
    }
}
private void dumpEvent(Event event){
    d_printStream.println("handler label=" + d_label);
    d_printStream.println("eventType=" + event.eventType());
    MessageIterator messageIterator = event.messageIterator();
    while (messageIterator.hasNext()){  
        Message message = messageIterator.next();
        d_printStream.println("messageType=" + message.messageType());
        d_printStream.println("CorrelationID=" + message.correlationID());
        try {
            message.print(d_printStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private void handleDataEvent(Event event, Session session){
    d_printStream.println("handleDataEvent: enter");
    dumpEvent(event);
    d_printStream.println("handleDataEvent: leave");
}

private void handleStatusEvent(Event event, Session session){
    d_printStream.println("handleStatusEvent: enter");
    dumpEvent(event);
    d_printStream.println("handleStatusEvent: leave");
}

private void handleOtherEvent(Event event, Session session){
    d_printStream.println("handleOtherEvent: enter");
    dumpEvent(event);
    d_printStream.println("handleOtherEvent: leave");
}

public class SubscriptionMultiple {

    public static void main(String[] args) throws Exception{
        SessionOptions sessionOptions = new SessionOptions();
        sessionOptions.setServerHost("localhost");
        sessionOptions.setServerPort(8194);
        Session session = new Session(sessionOptions,
                                      new SubscriptionEventHandler("myLabel",
                                                      System.out));

        if (!session.start()) {
            System.out.println("Could not start session.");
            System.exit(1);
        }
        if (!session.openService("//blp/mktdata")) {
            System.out.println("Could not open service " +
                                "+//blp/mktdata");
            System.exit(1);
        }
    }
}
SubscriptionList subscriptions = new SubscriptionList();
subscriptions.add(new Subscription("IBM US Equity",
    "LAST_TRADE",
    new CorrelationID(10)));
subscriptions.add(new Subscription("/ticker/GOOG US Equity",
    "BID,ASK,LAST_PRICE",
    new CorrelationID(20)));
subscriptions.add(new Subscription("MSFT US Equity",
    "LAST_PRICE",
    "interval=.5",
    new CorrelationID(30)));
subscriptions.add(new Subscription("/cusip/097023105?fields=LAST_PRICE&interval=5.0", //BA US Equity
    new CorrelationID(40)));

session.subscribe(subscriptions);

    // Wait for events
    Object object = new Object();
    synchronized (object) {
        object.wait();
    }
}
C.5.1 Multiple Subscription: Output

```
SuhandleStatusEvent: enter
  handler label=myLabel
eventType=SESSION_STATUS
messageType=SessionStarted
CorrelationID=null
SessionStarted = {
}  
handleStatusEvent: leave
handleStatusEvent: enter
  handler label=myLabel
eventType=SERVICE_STATUS
messageType=ServiceOpened
CorrelationID=Internal: 1
ServiceOpened = {
}  
handleStatusEvent: leave
handleStatusEvent: enter
  handler label=myLabel
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionFailure
CorrelationID=User: 30
SubscriptionFailure = {
  reason = {
    source = BBDB@n558
    errorCode = 2
    category = BAD_SEC
    description = Invalid security
  }
}  
handleStatusEvent: leave
handleStatusEvent: enter
  handler label=myLabel
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionStarted
CorrelationID=User: 10
SubscriptionStarted = {
}  
messageType=SubscriptionStarted
CorrelationID=User: 20
SubscriptionStarted = {
}  
messageType=SubscriptionStarted
CorrelationID=User: 40
SubscriptionStarted = {
}  
handleStatusEvent: leave
handleDataEvent: enter
  handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 20
```
MarketDataEvents = {
    LAST_PRICE = 343.32
    BID = 343.43
    ASK = 343.44
    VOLUME = 7283742
    HIGH = 345.05
    LOW = 340.11
    BEST_BID = 343.43
    BEST_ASK = 343.44
    LAST_TRADE = 343.32
    OPEN = 344.54
    PREV_SES_LASTPRICE = 348.67
    INDICATIVE_FAR = 343.16
    INDICATIVE_NEAR = 343.16
    VWAP = 342.842
    THEO_PRICE = 343.16
    LAST_ALL_SESSIONS = 344.2
    IMBALANCE_INDIC_RT = NOIM
    BID_ALL_SESSION = 343.4
    ASK_ALL_SESSION = 344.2
    TRADING_DT_REALTIME = 2009-01-29+00:00
    EQY_TURNOVER_REALTIME = 2.4559597933911133E9
    LAST_UPDATE_BID_RT = 21:00:00.000+00:00
    LAST_UPDATE_ASK_RT = 21:00:00.000+00:00
    TOT_CALL_VOLUME_CUR_DAY_RT = 3644
    TOT_PUT_VOLUME_CUR_DAY_RT = 3623
    TOT_OPT_VOLUME_CUR_DAY_RT = 7267
    PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
    IN_AUCTION_RT = false
    RT_API_MACHINE = n242
    ALL_PRICE_SIZE = 250
    ALL_PRICE = 344.2
    VOLUME_THEO = 732968
    BID_ASK_TIME = 21:00:00.000+00:00
    LAST_AT_TRADE_TDY = 0.0
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0.0
    HIGH_YLD_TDY = 0.0
    LOW_YLD_TDY = 0.0
    LAST_YLD_TDY = 0.0
    MID_TDY = 0.0
    SES_START = 14:30:00.000+00:00
    SES_END = 21:30:00.000+00:00
    RT_PX_CHG_NET_1D = -5.35
    RT_PX_CHG_PCT_1D = -1.5344
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
    OPEN_TDY = 344.54
    ASK_SIZE_TDY = 1
    BID_SIZE_TDY = 7
    VOLUME_TDY = 7283742
    LAST_PRICE_TDY = 343.32
}
BID_TDY = 343.43
ASK_TDY = 343.44
HIGH_TDY = 345.05
LOW_TDY = 340.11
BID_YLD_TDY = 0.0
ASK_YLD_TDY = 0.0
LAST2_PRICE = 340.54
LAST_DIR = 1
LAST2_DIR = -1
BID_DIR = 1
ASK_DIR = -1
BID2 = 343.4
ASK2 = 343.45
ASK_SIZE = 1
BID_SIZE = 7
TIME = 22:20:00.000+00:00
API_MACHINE = n242
TRADE_SIZE_ALL_SESSIONS_RT = 250
EID = 14005
IS_DELAYED_STREAM = false

chandleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 20
MarketDataEvents = {
    VOLUME = 7283742
    LAST_AT_TRADE_TDY = 0.0
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0.0
    HIGH_YLD_TDY = 0.0
    LOW_YLD_TDY = 0.0
    LAST_YLD_TDY = 0.0
    MID_TDY = 0.0
    RT_PX_CHG_NET_1D = -5.35
    RT_PX_CHG_PCT_1D = -1.5344
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
    OPEN_TDY = 344.54
    ASK_SIZE_TDY = 1
    BID_SIZE_TDY = 7
    VOLUME_TDY = 7283742
    LAST_PRICE_TDY = 343.32
    BID_TDY = 343.43
    ASK_TDY = 343.44
    HIGH_TDY = 345.05
    LOW_TDY = 340.11
    BID_YLD_TDY = 0.0
    ASK_YLD_TDY = 0.0
EID = 14005
IS DELAYED STREAM = false
}
handleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 10
MarketDataEvents = {
    LAST_PRICE = 92.51
    BID = 92.56
    ASK = 92.62
    VOLUME = 9233664
    HIGH = 94.58
    LOW = 92.02
    BEST_BID = 92.56
    BEST_ASK = 92.62
    LAST_TRADE = 92.51
    OPEN = 93.58
    PREV_SES_LAST_PRICE = 94.82
    IMBALANCE_ASK = 92.52
    ORDER_IMB_SELL_VOLUME = 34800.0
    VWAP = 93.2768
    THEO_PRICE = 92.52
    LAST_ALL_SESSIONS = 92.49
    IMBALANCE_INDIC_RT = SELL
    BID_ALL_SESSION = 92.31
    ASK_ALL_SESSION = 92.5
    TRADING_DT_REALTIME = 2009-01-29+00:00
    EQY_TURNOVER_REALTIME = 8.743154979367981E8
    LAST_UPDATE_BID_RT = 21:00:00.000+00:00
    LAST_UPDATE_ASK_RT = 21:00:00.000+00:00
    NYSE_LRP_HIGH_PRICE_RT = 93.63
    NYSE_LRP_LOW_PRICE_RT = 91.63
    NYSE_LRP_SEND_TIME_RT = 20:59:52.000+00:00
    TOT_CALL_VOLUME_CUR_DAY_RT = 4950
    TOT_PUT_VOLUME_CUR_DAY_RT = 7369
    TOT_OPT_VOLUME_CUR_DAY_RT = 12319
    PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
    IN AUCTION_RT = false
    RT_API_MACHINE = p065
    ALL_PRICE_SIZE = 200
    ALL_PRICE = 92.5
    VOLUME_THEO = 467100
    BID_ASK_TIME = 21:00:00.000+00:00
    LAST_AT_TRADE_TDY = 0.0
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0.0
    HIGH_YLD_TDY = 0.0
    LOW_YLD_TDY = 0.0
    MID_TDY = 0.0

```java
handleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 10
MarketDataEvents = {
    VOLUME = 9233664
    VWAP = 93.2764
    LAST_ALL_SESSIONS = 92.5
    BID_ALL_SESSION = 92.31
    ASK_ALL_SESSION = 92.5
    EQY_TURNOVER_REALTIME = 8.743154979367981E8
    ALL_PRICE_SIZE = 200
    ALL_PRICE = 92.5
    LAST_AT_TRADE_TDY = 0.0
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0.0
    HIGH_YLD_TDY = 0.0
    LOW_YLD_TDY = 0.0
```
LAST_YLD_TDY = 0.0
MID_TDY = 0.0
RT_PX_CHG_NET_1D = -2.31
RT_PX_CHG_PCT_1D = -2.43619
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 93.58
ASK_SIZE_TDY = 5
BID_SIZE_TDY = 1
VOLUME_TDY = 9233664
LAST_PRICE_TDY = 92.51
BID_TDY = 92.56
ASK_TDY = 92.62
HIGH_TDY = 94.58
LOW_TDY = 92.02
BID_YLD_TDY = 0.0
ASK_YLD_TDY = 0.0
TRADE_SIZE_ALL_SESSIONS_RT = 200
EID = 14003
IS_DELAYED_STREAM = false
}
handleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 40
MarketDataEvents = {
  LAST_PRICE = 40.71
  BID = 40.71
  ASK = 40.77
  VOLUME = 8446464
  HIGH = 42.76
  LOW = 40.37
  RT_OPEN_INTEREST = 7953467
  BEST_BID = 40.71
  BEST_ASK = 40.77
  LAST_TRADE = 40.71
  OPEN = 42.76
  PREV_SSES_LAST_PRICE = 43.24
  VWAP = 40.9212
  TRADING_DT_REALTIME = 2009-01-29+00:00
  EQY_TURNOVER_REALTIME = 3.45612128E8
  PREV_TRADING_DT_REALTIME = 2009-01-29+00:00
  RT_API_MACHINE = p164
  SES_START = 14:30:00.000+00:00
  SES_END = 21:30:00.000+00:00
  RT_PX_CHG_NET_1D = -2.53
  RT_PX_CHG_PCT_1D = -5.85106
  IND_BID_FLAG = false
  IND_ASK_FLAG = false
  OPEN_TDY = 42.76
  ASK_SIZE_TDY = 124
BID_SIZE_TDY = 228
VOLUME_TDY = 8446464
LAST_PRICE_TDY = 40.71
BID_TDY = 40.71
ASK_TDY = 40.77
HIGH_TDY = 42.76
LOW_TDY = 40.37
RT_PRICING_SOURCE = US
ASK_SIZE = 124
BID_SIZE = 228
TIME = 21:15:02.000+00:00
API_MACHINE = p164
EXCH_CODE_LAST = N
EXCH_CODE_BID = N
EXCH_CODE_ASK = N
EID = 14003
IS_DELAYED_STREAM = false
}
handleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 40
MarketDataEvents = {
    LAST_ALL_SESSIONS = 40.71
    BID_ALL_SESSION = 40.71
    ASK_ALL_SESSION = 40.77
    SES_START = 14:30:00.000+00:00
    SES_END = 21:30:00.000+00:00
    RT_PX_CHG_NET_1D = -2.53
    RT_PX_CHG_PCT_1D = -5.85106
    TIME = 21:15:02.000+00:00
    TRADE_SIZE_ALL_SESSIONS_RT = 0
    IS_DELAYED_STREAM = false
}
handleDataEvent: leave
C.6 Authorization by IP Address

// AuthorizationByIpAddress.java

package BloombergLP;

import java.io.IOException;
import java.util.ArrayList;
import com.bloomberglp.blpapi.CorrelationID;
import com.bloomberglp.blpapi.Element;
import com.bloomberglp.blpapi.Event;
import com.bloomberglp.blpapi.Message;
import com.bloomberglp.blpapi.MessageIterator;
import com.bloomberglp.blpapi.Request;
import com.bloomberglp.blpapi.Service;
import com.bloomberglp.blpapi.Session;
import com.bloomberglp.blpapi.SessionOptions;
import com.bloomberglp.blpapi.Identity;

public class AuthorizationByIpAddress {

    private static void dumpEvent(Event event) throws IOException{
        System.out.println("eventType=", event.eventType());
        MessageIterator messageIterator = event.messageIterator();
        while (messageIterator.hasNext()){
            Message message = messageIterator.next();
            System.out.println("messageType=", message.messageType());
            System.out.println("CorrelationID=", message.correlationID());
            message.print(System.out);
        }
    }

    private static boolean hasMessageType(Event event, String messageType) {
        MessageIterator messageIterator = event.messageIterator();
        while (messageIterator.hasNext()){
            Message message = messageIterator.next();
            if (message.messageType().equals(messageType)) {
                return true;
            }
        }
        return false;
    }

}
private static void printSecurityData(String security, int sequenceNumber, Element securityData)
{
    Element fieldData = securityData.getElement("fieldData");
    double px_last = fieldData.getElementAsFloat64("PX_LAST");
    String ds002 = fieldData.getElementAsString("DS002");
    double vwap_volume = fieldData.getElementAsFloat64("VWAP_VOLUME");

    // Individually output each value
    System.out.println("* security =" + security);
    System.out.println("* sequenceNumber =" + sequenceNumber);
    System.out.println("* px_last =" + px_last);
    System.out.println("* ds002 =" + ds002);
    System.out.println("* vwap_volume =" + vwap_volume);
    System.out.println("");
}

private static void handleResponseEvent(Event event, Identity identity) throws IOException {
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        Element ReferenceDataResponse = message.asElement();
        if (ReferenceDataResponse.hasElement("responseError")) {
            message.print(System.out);
            System.exit(1);
        }
        Element securityDataArray = ReferenceDataResponse.getElement("securityData");
        int numItems = securityDataArray.numValues();
        for (int i = 0; i < numItems; ++i) {
            Element securityData = securityDataArray.getValueAsElement(i);
            String security = securityData.getElementAsString("security");
            int sequenceNumber = securityData.getElementAsInt32("sequenceNumber");

            if (securityData.hasElement("securityError")) {
                Element securityError = securityData.getElement("securityError");
                System.out.println("* security =" + security);
                securityError.print(System.out);
                return;
            }
        }
    }
}
```java
ArrayList missingEntitlements = new ArrayList();
Element neededEntitlements =
securityData.hasElement("eidData")
    ? securityData.getElement("eidData") : null;
if (null == neededEntitlements) {
    System.out.println("no entitlements needed");
    System.out.println();
    printSecurityData(security, sequenceNumber, securityData);
} else if (identity.hasEntitlements(neededEntitlements,
        message.service(),
        missingEntitlements)) {
    System.out.println("user has the needed Entitlements for: "+ security);
    System.out.println("provide data to the requesting user");
    System.out.println();
    printSecurityData(security, sequenceNumber, securityData);
} else {
    System.out.println("user lacks entitlements for: "+ security);
    System.out.println("neededEntitlements = "
            + neededEntitlements);
    System.out.println("missingEntitlements = "
            + missingEntitlements);
    System.out.println();
    System.out.println("do not provide data to the requesting user");
}
}
}

private static void handleOtherEvent(Event event) throws Exception {
    System.out.println("EventType=" + event.eventType());
    MessageIterator iter = event.messageIterator();
    while (iter.hasNext()) {
        Message message = iter.next();
        System.out.println("correlationID="
                + message.correlationID());
        System.out.println("messageType=" + message.messageType());
        message.print(System.out);
        if (Event.EventType.Constants.SESSION_STATUS ==
            event.eventType().intValue() &&
            "SessionTerminated" ==
                message.messageType().toString()){
            System.out.println("Terminating: " +
                message.messageType());
            System.exit(1);
        }
    }
}
```
static private boolean handleAuthenticationResponseEvent(Event event) throws IOException{
    if (hasMessageType(event, "AuthorizationSuccess")) {
        System.out.println("Authorization OK");
        return true;
    } else if (hasMessageType(event, "AuthorizationFailure")) {
        System.out.println("Authorization Problem");
        dumpEvent(event);
    } else {
        System.out.println("Authorization: Other Problem");
        dumpEvent(event);
    }
    return false;
}

public static void main(String[] args) throws Exception{
    int uuid = uuid;
    String ipAddress = ipAddress;

    SessionOptions sessionOptions = new SessionOptions();
    sessionOptions.setServerHost("localhost"); //default
    sessionOptions.setServerPort(8194); //default
    Session session = new Session(sessionOptions);
    if (!session.start()) {
        System.out.println("Could not start session.");
        System.exit(1);
    }

    if (!session.openService("//blp/apiauth")) {
        System.out.println("Could not open service "+ "//blp/apiauth");
        System.exit(1);
    }
}
Service apiAuthSvc = session.getService("//blp/apiauth");

Request authorizationRequest =
    apiAuthSvc.createAuthorizationRequest();
authorizationRequest.set("uuid", uuid);
authorizationRequest.set("ipAddress", ipAddress);

Identity identity = session.createIdentity();
CorrelationID authorizationRequestID = new CorrelationID(10);

session.sendAuthorizationRequest(authorizationRequest,
    identity,
    authorizationRequestID);
System.out.println("sent Authorization Request using ipAddress");

// Wait for 'AuthorizationSuccess' message which indicates
// that 'identity' can be used.
for (boolean continueToLoop = true; continueToLoop; ) {
    Event event = session.nextEvent();
    //dumpEvent(event);
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.RESPONSE:
        if (!handleAuthenticationResponseEvent(event)) {
            System.out.println("Authorization Failed");
            System.exit(1);
        }
        continueToLoop = false;
        break;
    default:
        handleOtherEvent(event);
        break;
    }
}

if (!session.openService("//blp/refdata")) {
    System.out.println("Could not open service " + "+"";//blp/refdata");
    System.exit(1);
}
Service refDataSvc = session.getService("//blp/refdata");

Request request = refDataSvc.createRequest("ReferenceDataRequest");
request.append("securities", "VOD LN Equity");
request.append("fields", "PX_LAST");
request.append("fields", "DS002");
request.append("fields", "VWAP_VOLUME");
request.set("returnEids", true); // new

CorrelationID requestID = new CorrelationID(20);
session.sendRequest(request, requestID);
for (boolean continueToLoop = true; continueToLoop; ) {
    Event event = session.nextEvent();
    dumpEvent(event);
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.RESPONSE: // final event
        continueToLoop = false;              // fall through
        case Event.EventType.Constants.PARTIAL_RESPONSE:
            handleResponseEvent(event, identity); // new argument
            break;
        default:
            handleOtherEvent(event);
            break;
        }
    }
}
This section contains the following code examples:

- “RequestResponseParadigm” on page 260
- “Subscription Paradigm” on page 263
- “Asynchronous Event Handling” on page 269
- “Request Response Multiple” on page 274
- “Subscription Multiple” on page 278
using System;
using System.Collections.Generic;
using System.Text;

using CorrelationID = Bloomberglp.Blpapi.CorrelationID;
using Element = Bloomberglp.Blpapi.Element;
using Request = Bloomberglp.Blpapi.Request;
using Service = Bloomberglp.Blpapi.Service;
using Session = Bloomberglp.Blpapi.Session;
using SessionOptions = Bloomberglp.Blpapi.SessionOptions;

namespace RequestResponseParadigm
{
    class RequestResponseParadigm
    {
        static void Main(string[] args)
        {
            SessionOptions sessionOptions = new SessionOptions();
            sessionOptions.ServerHost = "localhost";
            sessionOptions.ServerPort = 8194;
            Session session = new Session(sessionOptions);
            if (!session.Start())
            {
                System.Console.WriteLine("Could not start session.");
                System.Environment.Exit(1);
            }
            if (!session.OpenService("//blp/refdata"))
            {
                System.Console.WriteLine("Could not open service "+"//blp/refdata");
                System.Environment.Exit(1);
            }
            CorrelationID requestID = new CorrelationID(1);
            Service refDataSvc = session.GetService("//blp/refdata");
            Request request = refDataSvc.CreateRequest("ReferenceDataRequest");
            request.Append("securities", "IBM US Equity");
            request.Append("fields", "PX_LAST");
            session.SendRequest(request, requestID);
bool continueToLoop = true;
while (continueToLoop)
{
    Event eventObj = session.NextEvent();
    switch (eventObj.Type)
    {
        case Event.EventType.RESPONSE: // final event
            continueToLoop = false;
            handleResponseEvent(eventObj);
            break;
        case Event.EventType.PARTIAL_RESPONSE:
            handleResponseEvent(eventObj);
            break;
        default:
            handleOtherEvent(eventObj);
            break;
    }
}

private static void handleResponseEvent(Event eventObj)
{
    System.Console.WriteLine("EventType =" + eventObj.Type);
    foreach (Message message in eventObj.GetMessages())
    {
        System.Console.WriteLine("correlationID=" +
            message.CorrelationID);
        System.Console.WriteLine("messageType =" +
            message.MessageType);
        message.Print(System.Console.Out);
    }
}

private static void handleOtherEvent(Event eventObj)
{
    System.Console.WriteLine("EventType= " + eventObj.Type);
    foreach (Message message in eventObj.GetMessages())
    {
        System.Console.WriteLine("correlationID=" +
            message.CorrelationID);
        System.Console.WriteLine("messageType=" +
            message.MessageType);
        message.Print(System.Console.Out);
        if (Event.EventTypeDef.SESSION_STATUS == eventObj.Type
            && message.MessageType.Equals("SessionTerminated"))
            System.Console.WriteLine("Terminating:" +
                message.MessageType);
        System.Environment.Exit(1);
    }
}
D.1.1 Request Response Paradigm Output

```csharp
eventType=SESSION_STATUS
correlationID=
messageType=SessionStarted
SessionStarted = {
}
eventType=SERVICE_STATUS
correlationID=Internal: 1
messageType=ServiceOpened
ServiceOpened = {
}
eventType =RESPONSE
correlationID=User: 1
messageType =ReferenceDataResponse
ReferenceDataResponse (choice) = {
  securityData[] = {
    securityData = {
      security = IBM US Equity
      sequenceNumber = 0
      fieldData = {
        PX_LAST = 91.84
      }
    }
  }
}
```
D.2 Subscription Paradigm

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using CorrelationID = Bloomberglp.Blpapi.CorrelationID;
using Session        = Bloomberglp.Blpapi.Session;
using SessionOptions = Bloomberglp.Blpapi.SessionOptions;
using Subscription   = Bloomberglp.Blpapi.Subscription;

namespace SubscriptionParadigm
{
    class SubscriptionParadigm
    {
        static void Main(string[] args)
        {
            SessionOptions sessionOptions = new SessionOptions();
            sessionOptions.ServerHost = "localhost";
            sessionOptions.ServerPort = 8194;
            Session session = new Session(sessionOptions);
            if (!session.Start())
            {
                System.Console.WriteLine("Could not start session.");
                System.Environment.Exit(1);
            }
            if (!session.OpenService("//blp/mktdata"))
            {
                System.Console.WriteLine("Could not open service "+"//blp/mktdata");
                System.Environment.Exit(1);
            }
            CorrelationID subscriptionID = new CorrelationID(2);
            List<Subscription> subscriptions = new List<Subscription>();
            subscriptions.Add(new Subscription("AAPL US Equity", "LAST_PRICE", subscriptionID));
            session.Subscribe(subscriptions);
        }
    }
}
```
int updateCount = 0;
while (true)
{
    Event eventObj = session.NextEvent();
    switch (eventObj.Type)
    {
        case Event.EventType.SUBSCRIPTION_DATA:
            handleDataEvent(eventObj, updateCount++);
            break;
        default:
            handleOtherEvent(eventObj);
            break;
    }
}

private static void handleDataEvent(Event eventObj, int updateCount)
{
    System.Console.WriteLine("EventType=" + eventObj.Type);
    System.Console.WriteLine("updateCount = " + updateCount);
    foreach (Message message in eventObj.GetMessages())
    {
        System.Console.WriteLine("correlationID = " +
            message.CorrelationID);  
        System.Console.WriteLine("messageType = " +
            message.MessageType);  
        message.Print(System.Console.Out);
    }
}

private static void handleOtherEvent(Event eventObj)
{
    System.Console.WriteLine("EventType=" + eventObj.Type);
    foreach (Message message in eventObj.GetMessages())
    {
        System.Console.WriteLine("correlationID=" +
            message.CorrelationID);
        System.Console.WriteLine("messageType=" +
            message.MessageType);
        message.Print(System.Console.Out);
        if (Event.EventType.SESSION_STATUS == eventObj.Type
            && message.MessageType.Equals("SessionTerminated"))
        {
            System.Console.WriteLine("Terminating: " +
                message.MessageType);
            System.Environment.Exit(1);
        }
    }
}
Subscription Paradigm Output

```
EventType=SESSION_STATUS
correlationID=
messageType=SessionStarted
SessionStarted = {
}
EventType=SERVICE_STATUS
correlationID=Internal: 1
messageType=ServiceOpened
ServiceOpened = {
}
EventType=SUBSCRIPTION_STATUS
correlationID=User: 2
messageType=SubscriptionStarted
SubscriptionStarted = {
}
EventType=SUBSCRIPTION_DATA
updateCount = 0
correlationID = User: 2
messageType = MarketDataEvents
MarketDataEvents = {
    LAST_PRICE = 90.89
    BID = 90.88
    ASK = 90.9
    VOLUME = 14304168
    HIGH = 93.62
    LOW = 90.6
    BEST_BID = 90.88
    BEST_ASK = 90.9
    LAST_TRADE = 90.89
    OPEN = 92.6
    PREV_SES_LAST_PRICE = 93
    INDICATIVE_FAR = 92.62
    INDICATIVE_NEAR = 92.62
    IMBALANCE_BID = 92.6
    VWAP = 91.9119
    LAST_ALL_SESSIONS = 90.89
    IMBALANCE_INDIC_RT = BUY
    BID_ALL_SESSION = 90.88
    ASK_ALL_SESSION = 90.9
    TRADING_DT_REALTIME = 2009-01-30+00:00
    EQY_TURNOVER_REALTIME = 1294308731.96565
    LAST_UPDATE_BID_RT = 18:45:46.000+00:00
    LAST_UPDATE_ASK_RT = 18:45:46.000+00:00
    TOT_CALL_VOLUME_CUR_DAY_RT = 12783
    TOT_PUT_VOLUME_CUR_DAY_RT = 17211
    TOT_OPT_VOLUME_CUR_DAY_RT = 29994
    PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
    IN_AUCTION_RT = false
    RT_API_MACHINE = p060
    ALL_PRICE_SIZE = 100
    ALL_PRICE = 90.89
```
BID_ASK_TIME = 18:45:46.000+00:00
LAST_AT_TRADE_TDY = 0
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0
HIGH_YLD_TDY = 0
LOW_YLD_TDY = 0
LAST_YLD_TDY = 0
MID_TDY = 0
SIZE_LAST_TRADE_TDY = 100
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
RT_PX_CHG_NET_1D = -2.11
RT_PX_CHG_PCT_1D = -2.26882
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 92.6
ASK_SIZE_TDY = 19
BID_SIZE_TDY = 5
VOLUME_TDY = 14304168
LAST_PRICE_TDY = 90.89
BID_TDY = 90.88
ASK_TDY = 90.9
HIGH_TDY = 93.62
LOW_TDY = 90.6
BID_YLD_TDY = 0
ASK_YLD_TDY = 0
LAST2_PRICE = 90.89
LAST_DIR = 1
LAST2_DIR = 1
BID_DIR = -1
ASK_DIR = 1
BID2 = 90.88
ASK2 = 90.9
SIZE_LAST_TRADE = 100
ASK_SIZE = 19
BID_SIZE = 5
TIME = 18:45:45.000+00:00
API_MACHINE = p060
TRADE_SIZE_ALL_SESSIONS_RT = 100
EID = 14005
IS_DELAYED_STREAM = false
EventType=SUBSCRIPTION_DATA
updateCount = 1
correlationID = User: 2
messageType   = MarketDataEvents
MarketDataEvents = {
    LAST_PRICE = 90.89
    BID = 90.88
    ASK = 90.9
    VOLUME = 14304168
    HIGH = 93.62
    LOW = 90.6
}
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEST_BID</td>
<td>90.88</td>
</tr>
<tr>
<td>BEST_ASK</td>
<td>90.9</td>
</tr>
<tr>
<td>LAST_TRADE</td>
<td>90.89</td>
</tr>
<tr>
<td>VWAP</td>
<td>91.6348</td>
</tr>
<tr>
<td>LAST_ALL_SESSIONS</td>
<td>90.89</td>
</tr>
<tr>
<td>BID_ALL_SESSION</td>
<td>90.88</td>
</tr>
<tr>
<td>ASK_ALL_SESSION</td>
<td>90.9</td>
</tr>
<tr>
<td>EQY_TURNOVER_REALTIME</td>
<td>1294308731.96565</td>
</tr>
<tr>
<td>LAST_UPDATE_BID_RT</td>
<td>18:45:46.000+00:00</td>
</tr>
<tr>
<td>LAST_UPDATE_ASK_RT</td>
<td>18:45:46.000+00:00</td>
</tr>
<tr>
<td>TOT_CALL_VOLUME_CUR_DAY_RT</td>
<td>12783</td>
</tr>
<tr>
<td>TOT_PUT_VOLUME_CUR_DAY_RT</td>
<td>17211</td>
</tr>
<tr>
<td>TOT_OPT_VOLUME_CUR_DAY_RT</td>
<td>29994</td>
</tr>
<tr>
<td>PUT_CALL_VOLUME_RATIO_CUR_DAY_RT</td>
<td>1</td>
</tr>
<tr>
<td>IN_AUCTION_RT</td>
<td>false</td>
</tr>
<tr>
<td>ALL_PRICE_SIZE</td>
<td>100</td>
</tr>
<tr>
<td>ALL_PRICE</td>
<td>90.89</td>
</tr>
<tr>
<td>BID_ASK_TIME</td>
<td>18:45:46.000+00:00</td>
</tr>
<tr>
<td>LAST_AT_TRADE_TDY</td>
<td>0</td>
</tr>
<tr>
<td>SIZE_LAST_AT_TRADE_TDY</td>
<td>0</td>
</tr>
<tr>
<td>OPEN_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>HIGH_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>LOW_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>LAST_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>MID_TDY</td>
<td>0</td>
</tr>
<tr>
<td>SIZE_LAST_TRADE_TDY</td>
<td>100</td>
</tr>
<tr>
<td>RT_PX_CHG_NET_1D</td>
<td>-2.11</td>
</tr>
<tr>
<td>RT_PX_CHG_PCT_1D</td>
<td>-2.26882</td>
</tr>
<tr>
<td>IND_BID_FLAG</td>
<td>false</td>
</tr>
<tr>
<td>IND_ASK_FLAG</td>
<td>false</td>
</tr>
<tr>
<td>OPEN_TDY</td>
<td>92.6</td>
</tr>
<tr>
<td>ASK_SIZE_TDY</td>
<td>19</td>
</tr>
<tr>
<td>BID_SIZE_TDY</td>
<td>5</td>
</tr>
<tr>
<td>VOLUME_TDY</td>
<td>14304168</td>
</tr>
<tr>
<td>LAST_PRICE_TDY</td>
<td>90.89</td>
</tr>
<tr>
<td>BID_TDY</td>
<td>90.88</td>
</tr>
<tr>
<td>ASK_TDY</td>
<td>90.9</td>
</tr>
<tr>
<td>HIGH_TDY</td>
<td>93.62</td>
</tr>
<tr>
<td>LOW_TDY</td>
<td>90.6</td>
</tr>
<tr>
<td>BID_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>ASK_YLD_TDY</td>
<td>0</td>
</tr>
<tr>
<td>LAST2_PRICE</td>
<td>90.89</td>
</tr>
<tr>
<td>LAST_DIR</td>
<td>1</td>
</tr>
<tr>
<td>LAST2_DIR</td>
<td>1</td>
</tr>
<tr>
<td>BID_DIR</td>
<td>-1</td>
</tr>
<tr>
<td>ASK_DIR</td>
<td>1</td>
</tr>
<tr>
<td>BID2</td>
<td>90.88</td>
</tr>
<tr>
<td>ASK2</td>
<td>90.9</td>
</tr>
<tr>
<td>SIZE_LAST_TRADE</td>
<td>100</td>
</tr>
<tr>
<td>ASK_SIZE</td>
<td>19</td>
</tr>
<tr>
<td>BID_SIZE</td>
<td>5</td>
</tr>
</tbody>
</table>
```csharp
TIME = 18:45:45.000+00:00
TRADE_SIZE_ALL_SESSIONS_RT = 100
EID = 14005
IS_DELAYED_STREAM = false
```
// AsynchronousEventHandling.cs

using System;
using System.Collections.Generic;
using System.Text;
using CorrelationID = BloombergLP.Blpapi.CorrelationID;
using Event = BloombergLP.Blpapi.Event;
using EventHandler = BloombergLP.Blpapi.EventHandler;
using Message = BloombergLP.Blpapi.Message;
using Request = BloombergLP.Blpapi.Request;
using Service = BloombergLP.Blpapi.Service;
using Session = BloombergLP.Blpapi.Session;
using SessionOptions = BloombergLP.Blpapi.SessionOptions;

namespace BloombergLP
{
    class AsynchronousEventHandling
    {
        static void Main(string[] args)
        {
            SessionOptions sessionOptions = new SessionOptions();
            sessionOptions.ServerHost = "localhost";
            sessionOptions.ServerPort = 8194;
            Session session = new Session(sessionOptions,
                new EventHandler(ProcessEvent));
            session.StartAsync();
            // Wait for events
            Object obj = new Object();
            lock (obj)
            {
                System.Threading.Monitor.Wait(obj);
            }
        }
    }

    static void dumpEvent(Event eventObj)
    {
        System.Console.WriteLine("eventType=");
        foreach (Message message in eventObj.GetMessages())
        {
            System.Console.WriteLine("messageType=" + message.MessageType);
            System.Console.WriteLine("CorrelationID=" + message.CorrelationID);
        }
    }
}

try
{
    message.Print(System.Console.Out);
}
catch (System.IO.IOException e)
{
    System.Console.WriteLine(e);
}
}
}

static public void ProcessEvent(Event eventObj, Session session)
{
    switch (eventObj.Type)
    {
    case Event.EventType.SESSION_STATUS:
    {
        foreach (Message message in eventObj.GetMessages())
        {
            if (message.MessageType.Equals("SessionStarted"))
            {
                try
                {
                    session.OpenServiceAsync("//blp/refdata",
                                               new CorrelationID(99));
                }
                catch (Exception)
                {
                    System.Environment.Exit(1);
                }
            }
            else
            {
                System.Console.Error.WriteLine("Could not start session.");
                System.Environment.Exit(1);
            }
        }
    }
    break;
    }
}
case Event.Event.Type.SERVICE_STATUS:
{
    foreach (Message message in eventObj.GetMessages())
    {
        if (message.CorrelationID.Value == 99 && message.MessageType.Equals("ServiceOpened"))
        {
            //Construct and issue a Request
            Service service = session.GetService("/blp/refdata");
            Request request = service.CreateRequest("ReferenceDataRequest");
            request.Append("securities", "IBM US Equity");
            request.Append("fields", "PX_LAST");
            try
            {
                session.SendRequest(request,
                                    new CorrelationID(86));
            }
            catch (Exception)
            {
                System.Console.Error.WriteLine("Could not send request");
                System.Environment.Exit(1);
            }
        }
        else
        {
            System.Console.WriteLine("Unexpected SERVICE_STATUS message:");
            try
            {
                message.Print(System.Console.Error);
            }
            catch (Exception e)
            {
                System.Console.WriteLine(e);
            }
        }
    }
    break;
}
case Event.EventType.PARTIAL_RESPONSE:
{
    dumpEvent(eventObj); // Handle Partial Response
    break;
}

case Event.EventType.RESPONSE:
{
    dumpEvent(eventObj); // Handle final response
    // Now, the example is complete. Shut it down.
    try
    {
        session.Stop(Session.StopOption.ASYNC);
    }
    catch (System.Threading.ThreadInterruptedException e)
    {
        System.Console.WriteLine(e);
    }
    System.Console.Error.WriteLine("terminate process from handler");
    System.Environment.Exit(0);
    break;
}
default:
{
    break;
}

case Event.EventType.RESPONSE:
{
    dumpEvent(eventObj); // Handle final response
    System.Console.WriteLine("unexpected Event");
    dumpEvent(eventObj);
    System.Environment.Exit(1);
    break;
}
D.3.1 Asynchronous Event Handling: Output

```csharp
eventType=RESPONSE
messageType=ReferenceDataResponse
CorrelationID=User: 86
ReferenceDataResponse (choice) = {
    securityData[] = {
        securityData = {
            security = IBM US Equity
            sequenceNumber = 0
            fieldData = {
                PX_LAST = 91.85
            }
        }
    }
}
```
using System;
using System.Collections.Generic;
using System.Text;
using CorrelationID = Bloomberglp.Blpapi.CorrelationID;
using Element = Bloomberglp.Blpapi.Element;
using Request = Bloomberglp.Blpapi.Request;
using Service = Bloomberglp.Blpapi.Service;
using Session = Bloomberglp.Blpapi.Session;
using SessionOptions = Bloomberglp.Blpapi.SessionOptions;

namespace RequestResponseMultiple
{
    class RequestResponseMultiple
    {
        static void Main(string[] args)
        {
            SessionOptions sessionOptions = new SessionOptions();
            sessionOptions.ServerHost = "localhost";
            sessionOptions.ServerPort = 8194;
            Session session = new Session(sessionOptions);
            if (!session.Start())
            {
                System.Console.WriteLine("Could not start session.");
                System.Environment.Exit(1);
            }
            if (!session.OpenService("//blp/refdata"))
            {
                System.Console.WriteLine("Could not open service "+" //blp/refdata");
                System.Environment.Exit(1);
            }
            Service refDataSvc = session.GetService("//blp/refdata");
            Request request = refDataSvc.CreateRequest("ReferenceDataRequest");
            request.GetElement("securities").AppendValue("AAPL US Equity");
            request.GetElement("securities").AppendValue("IBM US Equity");
            request.GetElement("securities").AppendValue("BLAHBLAHBLAH US Equity");
            request.GetElement("fields").AppendValue("PX_LAST");
            request.GetElement("fields").AppendValue("DS002");
            request.GetElement("fields").AppendValue("VWAP_VOLUME");
            session.SendRequest(request, new CorrelationID(1));
        }
    }
}
bool continueToLoop = true;
while (continueToLoop)
{
    Event eventObj = session.NextEvent();
    switch (eventObj.Type)
    {
        case Event.EventType.RESPONSE:   // final response
            continueToLoop = false;
            handleResponseEvent(eventObj);
            break;
        case Event.EventType.PARTIAL_RESPONSE:  
            handleResponseEvent(eventObj);
            break;
        default:
            handleOtherEvent(eventObj);
            break;
    }
}

private static void handleResponseEvent(Event eventObj)
{
    foreach (Message message in eventObj.GetMessages())
    {
        Element ReferenceDataResponse = message.AsElement;
        if (ReferenceDataResponse.HasElement("responseError"))
        {
            System.Environment.Exit(1);
        }
        Element securityDataArray =
            ReferenceDataResponse.GetElement("securityData");
        int numItems = securityDataArray.NumValues;
        for (int i = 0; i < numItems; ++i)
        {
            Element securityData =
                securityDataArray.GetValueAsElement(i);
            String security =
                securityData.GetElementAsString("security");
            int sequenceNumber =
                securityData.GetElementAsInt32("sequenceNumber");
            if (securityData.HasElement("securityError"))
            {
                Element securityError =
                    securityData.GetElement("securityError");
                System.Console.WriteLine("* security      =" +
                    security);
                Element securityError =
                    securityData.GetElement("securityError");
                securityError.Print(System.Console.Out);
                return;
            }
        }
    }
else
{
    Element fieldData =
        securityData.GetElement("fieldData");
    double px_last =
        fieldData.GetElementAsFloat64("PX_LAST");
    String ds002 =
        fieldData.GetElementAsString("DS002");
    double vwap_volume =
        fieldData.GetElementAsFloat64("VWAP_VOLUME");

    // Individually output each value
    System.Console.WriteLine("* security =" +
                           security);
    System.Console.WriteLine("* sequenceNumber=" +
                           sequenceNumber);
    System.Console.WriteLine("* px_last =" +
                           px_last);
    System.Console.WriteLine("* ds002 =" +
                           ds002);
    System.Console.WriteLine("* vwap_volume =" +
                           vwap_volume);
    System.Console.WriteLine();
}
}

private static void handleOtherEvent(Event eventObj)
{
    System.Console.WriteLine("EventType=" + eventObj.Type);
    foreach (Message message in eventObj.GetMessages())
    {
        System.Console.WriteLine("correlationID=" +
                               message.CorrelationID);
        System.Console.WriteLine("messageType=" +
                               message.MessageType);
        message.Print(System.Console.Out);
        if (Event.EventType.SESSION_STATUS == eventObj.Type
            && message.MessageType.Equals("SessionTerminated"))
        {
            System.Console.WriteLine("Terminating: " +
                                   message.MessageType);
            System.Environment.Exit(1);
        }
    }
}
}
D.4.1 Request Response Multiple: Output

```plaintext
EventType=SESSION_STATUS
    correlationID=
    messageType=SessionStarted
    SessionStarted = {

EventType=SERVICE_STATUS
    correlationID=Internal: 1
    messageType=ServiceOpened
    ServiceOpened = {
        * security = AAPL US Equity
        * sequenceNumber=0
        * px_last = 90.95
        * ds002 = APPLE INC
        * vwap_volume = 14300635

        * security = IBM US Equity
        * sequenceNumber=1
        * px_last = 92.04
        * ds002 = INTL BUSINESS MACHINES CORP
        * vwap_volume = 4661754

        * security = BLAHLAHBLAHL US Equity
    } securityError = {
        source = 236::bbdbs2
        code = 15
        category = BAD_SEC
        message = Unknown/Invalid security [nid:236]
        subcategory = INVALID_SECURITY
    }
```

namespace SubscriptionMultiple
{
    class SubscriptionEventHandler {
        private String d_label;
        private TextWriter d_printStream;

        // CREATORS
        public SubscriptionEventHandler(String label, TextWriter printStream)
        {
            d_label = label;
            d_printStream = printStream;
        }

        // MANIPULATORS
        public void ProcessEvent(Event eventObj, Session session)
        {
            switch (eventObj.Type)
            {
                case Event.EventType.SUBSCRIPTION_DATA:
                    handleDataEvent(eventObj, session);
                    break;
                case Event.EventType.SESSION_STATUS:
                case Event.EventType.SERVICE_STATUS:
                case Event.EventType.SUBSCRIPTION_STATUS:
                    handleStatusEvent(eventObj, session);
                    break;
                default:
                {
                    handleOtherEvent(eventObj, session);
                    break;
                }
            }
        }
    }
}
private void dumpEvent(Event eventObj)
{
    d_printStream.WriteLine("handler label=" + d_label);
    d_printStream.WriteLine("eventType=" + eventObj.Type);
    foreach (Message message in eventObj.GetMessages())
    {
        d_printStream.WriteLine("messageType=" + message.MessageType);
        d_printStream.WriteLine("CorrelationID=" + message.CorrelationID);
        try
        {
            message.Print(d_printStream);
        }
        catch (IOException e)
        {
            System.Console.WriteLine(e);
        }
    }
}

private void handleDataEvent(Event eventObj, Session session)
{
    d_printStream.WriteLine("handleDataEvent: enter");
    dumpEvent(eventObj);
    d_printStream.WriteLine("handleDataEvent: leave");
}

private void handleStatusEvent(Event eventObj, Session session)
{
    d_printStream.WriteLine("handleStatusEvent: enter");
    dumpEvent(eventObj);
    d_printStream.WriteLine("handleStatusEvent: leave");
}

private void handleOtherEvent(Event eventObj, Session session)
{
    d_printStream.WriteLine("handleOtherEvent: enter");
    dumpEvent(eventObj);
    d_printStream.WriteLine("handleOtherEvent: leave");
}

class SubscriptionMultiple
{
    static void Main(string[] args)
    {
        SessionOptions sessionOptions = new SessionOptions();
        sessionOptions.ServerHost = "localhost";
        sessionOptions.ServerPort = 8194;
        Session session = new Session(sessionOptions, new EventHandler(
    }
}
if (!session.Start())
{
    System.Console.WriteLine("Could not start session.");
    System.Environment.Exit(1);
}
if (!session.OpenService("/blp/mktdata"))
{
    System.Console.WriteLine("Could not open service "+"/blp/mktdata");
    System.Environment.Exit(1);
}

List<Subscription> subscriptions = new List<Subscription>();
subscriptions.Add(new Subscription("IBM US Equity",
    "LAST_TRADE",
    new CorrelationID(10)));
subscriptions.Add(new Subscription("/ticker/GOOG US Equity",
    "BID,ASK,LAST_PRICE",
    new CorrelationID(20)));
subscriptions.Add(new Subscription("MSFTT US Equity",
    "LAST_PRICE",
    "interval=.5",
    new CorrelationID(30)));
subscriptions.Add(new Subscription( //BA US Equity
    "/cusip/097023105?fields=LAST_PRICE&interval=5.0",
    new CorrelationID(40)));

session.Subscribe(subscriptions);

// Wait for events
Object obj = new Object();
lock (obj)
{
    System.Threading.Monitor.Wait(obj);
}
}
D.5.1 Multiple Subscription: Output

```c
handleStatusEvent: enter
handler label=myLabel
eventType=SESSION_STATUS
messageType=SessionStarted
CorrelationID=
SessionStarted = {
}
handleStatusEvent: leave
handleStatusEvent: enter
handler label=myLabel
eventType=SERVICE_STATUS
messageType=ServiceOpened
CorrelationID=Internal: 1
ServiceOpened = {
}
handleStatusEvent: leave
handleStatusEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionFailure
CorrelationID=User: 30
SubscriptionFailure = {
    reason = {
        source = BBDB@n558
        errorCode = 2
        category = BAD_SEC
        description = Invalid security
    }
}
handleStatusEvent: leave
handleStatusEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_STATUS
messageType=SubscriptionStarted
CorrelationID=User: 10
SubscriptionStarted = {
}
messageType=SubscriptionStarted
CorrelationID=User: 20
SubscriptionStarted = {
}
messageType=SubscriptionStarted
CorrelationID=User: 40
SubscriptionStarted = {
}
```
handleStatusEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 20
MarketDataEvents = {
    LAST_PRICE = 340.7
    BID = 340.74
    ASK = 340.92
    VOLUME = 2630520
    HIGH = 348.8
    LOW = 337.62
    BEST_BID = 340.74
    BEST_ASK = 340.92
    LAST_TRADE = 340.7
    OPEN = 344.69
    PREV_SES_LAST_PRICE = 343.32
    INDICATIVE_FAR = 344.69
    INDICATIVE_NEAR = 344.69
    IMBALANCE_ASK = 344.76
    VWAP = 341.6714
    LAST_ALL_SESSIONS = 340.7
    IMBALANCE_INDIC_RT = SELL
    BID_ALL_SESSION = 340.74
    ASK_ALL_SESSION = 340.92
    TRADING_DT_REALTIME = 2009-01-30+00:00
    EQY_TURNOVER_REALTIME = 891123786.45166
    LAST_UPDATE_BID_RT = 18:46:07.000+00:00
    LAST_UPDATE_ASK_RT = 18:46:09.000+00:00
    TOT_CALL_VOLUME_CUR_DAY_RT = 2146
    TOT_PUT_VOLUME_CUR_DAY_RT = 2887
    TOT_OPT_VOLUME_CUR_DAY_RT = 5033
    PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
    IN_AUCTION_RT = false
    RT_API_MACHINE = p060
    ALL_PRICE_SIZE = 300
    ALL_PRICE = 340.7
    BID_ASK_TIME = 18:46:09.000+00:00
    LAST_AT_TRADE_TDY = 0
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0
    HIGH_YLD_TDY = 0
    LOW_YLD_TDY = 0
    LAST_YLD_TDY = 0
    MID_TDY = 0
    SIZE_LAST_TRADE_TDY = 300
    SES_START = 14:30:00.000+00:00
    SES_END = 21:30:00.000+00:00
    RT_PX_CHG_NET_1D = -2.62
    RT_PX_CHG_PCT_1D = -0.763135
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
OPEN_TDY = 344.69
ASK_SIZE_TDY = 3
BID_SIZE_TDY = 3
VOLUME_TDY = 2630520
LAST_PRICE_TDY = 340.7
BID_TDY = 340.74
ASK_TDY = 340.92
HIGH_TDY = 348.8
LOW_TDY = 337.62
BID_YLD_TDY = 0
ASK_YLD_TDY = 0
LAST2_PRICE = 340.77
LAST_DIR = -1
LAST2_DIR = -1
BID_DIR = 1
ASK_DIR = -1
BID2 = 340.74
ASK2 = 340.92
SIZE_LAST_TRADE = 300
ASK_SIZE = 3
BID_SIZE = 3
TIME = 18:46:02.000+00:00
API_MACHINE = p060
TRADE_SIZE_ALL_SESSIONS_RT = 300
EID = 14005
IS_DELAYED_STREAM = false
}
handleDataEvent: leave
handleDataEvent: enter
handler label=myLabel
eventType=SUBSCRIPTION_DATA
messageType=MarketDataEvents
CorrelationID=User: 10
MarketDataEvents = {
    LAST_PRICE = 91.88
    BID = 91.85
    ASK = 91.88
    VOLUME = 4625564
    HIGH = 93.48
    LOW = 91.56
    BEST_BID = 91.85
    BEST_ASK = 91.88
    LAST_TRADE = 91.88
    OPEN = 92.23
    PREV_SES_LAST_PRICE = 92.51
    VWAP = 92.5054
    THEO_PRICE = 0
    LAST_ALL_SESSIONS = 91.88
    IMBALANCE_INDIC_RT = NOIM
    BID_ALL_SESSION = 91.85
    ASK_ALL_SESSION = 91.88
    TRADING_DT_REALTIME = 2009-01-30+00:00
    EQY_TURNOVER_REALTIME = 426434047.387161
Bloomberg

FINANCIAL_STATUS_INDICATOR_RT = 0
LAST_UPDATE_BID_RT = 18:46:09.000+00:00
LAST_UPDATE_ASK_RT = 18:46:09.000+00:00
NYSE_LRP_HIGH_PRICE_RT = 92.85
NYSE_LRP_LOW_PRICE_RT = 90.85
NYSE_LRP_SEND_TIME_RT = 18:46:08.000+00:00
TOT_CALL_VOLUME_CUR_DAY_RT = 1507
TOT_PUT_VOLUME_CUR_DAY_RT = 2122
TOT_OPT_VOLUME_CUR_DAY_RT = 3629
PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
IN_AUCTION_RT = false
RT_API_MACHINE = n160
ALL_PRICE_SIZE = 100
ALL_PRICE = 91.88
VOLUME_THEO = 0
BID_ASK_TIME = 18:46:09.000+00:00
LAST_AT_TRADE_TDY = 0
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0
HIGH_YLD_TDY = 0
LOW_YLD_TDY = 0
LAST_YLD_TDY = 0
MID_TDY = 0
SIZE_LAST_TRADE_TDY = 100
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
RT_PX_CHG_NET_1D = -0.6299
RT_PX_CHG_PCT_1D = -0.680898
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 92.23
ASK_SIZE_TDY = 1
BID_SIZE_TDY = 3
VOLUME_TDY = 4625564
LAST_PRICE_TDY = 91.88
BID_TDY = 91.85
ASK_TDY = 91.88
HIGH_TDY = 93.48
LOW_TDY = 91.56
BID_YLD_TDY = 0
ASK_YLD_TDY = 0
LAST2_PRICE = 91.87
LAST_DIR = 1
LAST2_DIR = 1
BID_DIR = 1
ASK_DIR = 1
BID2 = 91.85
ASK2 = 91.88
SIZE_LAST_TRADE = 100
ASK_SIZE = 1
BID_SIZE = 3
TIME = 18:46:09.000+00:00
API_MACHINE = n160
TRADE_SIZE_ALL_SESSIONS_RT = 100
EID = 14003
IS_DELAYED_STREAM = false
}
This section contains the following code examples:

- “RequestResponseParadigm” on page 287
- “Subscription Paradigm” on page 290
- “Asynchronous Event Handling” on page 295
- “Request Response Multiple” on page 299
- “Subscription Multiple” on page 303

**Note:** These examples use assert statements to make manifest the program state at various key points. Follow your organization’s guidelines for best practices on the use of assert statements in production code.
// RequestResponseParadigm.cpp

#include <blpapi_correlationid.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <iostream>
#include <cstring> // for strcmp(3C)

using namespace BloombergLP;
using namespace blpapi;

static void handleResponseEvent(const Event& event)
{
    std::cout << "EventType =" << event.eventType() << std::endl;
    MessageIterator iter(event);
    while (iter.next()) {
        Message message = iter.message();
        std::cout << "correlationId =" << message.correlationId() << std::endl;
        std::cout << "messageType =" << message.messageType() << std::endl;
        message.print(std::cout);
    }
}

static void handleOtherEvent(const Event& event)
{
    std::cout << "EventType =" << event.eventType() << std::endl;
    MessageIterator iter(event);
    while (iter.next()) { 
        Message message = iter.message();
        std::cout << "correlationId =" << message.correlationId() << std::endl;
        std::cout << "messageType =" << message.messageType() << std::endl;
    }
}
message.print(std::cout);
if (Event::SESSION_STATUS == event.eventType() &&
    0 == ::strcmp("SessionTerminated", message.messageType().string())) {
    std::cout << "Terminating: "
               << message.messageType()
               << std::endl;
    ::exit(1);
}
}
}

int main() {
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);
    
    Session session(sessionOptions);  // Establish session
    // Start Session
    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return 1;
    }
    if (!session.openService("//blp/refdata")){
        std::cerr << "Failed to open service //blp/refdata." << std::endl;
        return 1;
    }
    CorrelationId requestId(1);
    Service refDataSvc = session.getService("//blp/refdata");
    
    Request request = refDataSvc.createRequest("ReferenceDataRequest");
    request.append("securities", "IBM US Equity");
    request.append("fields", "PX_LAST");
    session.sendRequest(request, requestId);
bool continueToLoop = true;
while (continueToLoop) {
    Event event = session.nextEvent();
    switch (event.eventType()) {
        case Event::RESPONSE:      // final event
            continueToLoop = false;  // fall through
            case Event::PARTIAL_RESPONSE:
                handleResponseEvent(event);
                break;
        default:
            handleOtherEvent(event);
            break;
    }
}

session.stop();
return 0;

---

Request Response Paradigm Output

```plaintext
EventType=2
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=SessionStarted
SessionStarted = {
}

EventType=9
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=ServiceOpened
ServiceOpened = {
}

EventType=5
correlationId=[ valueType=INT classId=0 value=1 ]
messageType=ReferenceDataResponse
ReferenceDataResponse = {
    securityData[] =
        securityData = {
            security = IBM US Equity
eidData[] =

            fieldExceptions[] =

            sequenceNumber = 0
            fieldData = {
                PX_LAST = 92.510000
                ...
            }
        }
}
```
E.2 Subscription Paradigm

```cpp
// SubscriptionParadigm.cpp

#include <blpapi_correlationid.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>
#include <blpapi.subscriptionlist.h>

#include <iostream>
using namespace BloombergLP;
using namespace blpapi;

static void handleDataEvent(const Event& event, int updateCount) {
    std::cout << "EventType=
" << event.eventType() << std::endl;
    std::cout << "updateCount = " << updateCount << std::endl;
    MessageIterator iter(event);
    while (iter.next()) {
        Message message = iter.message();
        std::cout << "correlationId = "
                   << message.correlationId() << std::endl;
        std::cout << "messageType = "
                   << message.messageType() << std::endl;
        message.print(std::cout);
    }
}

static void handleOtherEvent(const Event& event) {
    std::cout << "EventType=
" << event.eventType() << std::endl;
}
```
MessageIterator iter(event);
while (iter.next()) {
    Message message = iter.message();
    std::cout << "correlationId="
               << message.correlationId()
               << std::endl;
    std::cout << "messageType="
               << message.messageType()
               << std::endl;
    message.print(std::cout);
    if (Event::SESSION_STATUS == event.eventType() 
        && 0 == ::strcmp("SessionTerminated", 
                         message.messageType().string())) {
        std::cout << "Terminating: "
                   << message.messageType()
                   << std::endl;
        ::exit(1);
    }
}

int main(int argc, char **argv)
{
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);
    Session session(sessionOptions);

    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return 1;
    }

    if (!session.openService("//blp/mktdata")) {
        std::cerr << "Failed to open //blp/mktdata" << std::endl;
        return 1;
    }
}
CorrelationId subscriptionId((long long)2);
SubscriptionList subscriptions;
subscriptions.add("AAPL US Equity",
   "LAST_PRICE",
   ":",
   subscriptionId);
session.subscribe(subscriptions);

int updateCount = 0;
while (true) {
    Event event = session.nextEvent();
    switch (event.eventType()) {
        case Event::SUBSCRIPTION_DATA: handleDataEvent(event, updateCount++);
        break;
        default: handleOtherEvent(event);
        break;
    }
    return 0;
}
Subscription Paradigm Output

```cpp
EventType=2
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=SessionStarted
SessionStarted = {
}

EventType=9
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=ServiceOpened
ServiceOpened = {
}

EventType=3
correlationId=[ valueType=INT classId=0 value=2 ]
messageType=SubscriptionStarted
SubscriptionStarted = {
    exceptions[] =
}

EventType=8
updateCount = 0
correlationId = [ valueType=INT classId=0 value=2 ]
messageType = MarketDataEvents
MarketDataEvents = {
    LAST_PRICE = 93.000000
    BID = 92.920000
    ASK = 92.950000
    VOLUME = 21170839
    HIGH = 94.340000
    LOW = 92.600000
    RT_OPEN_INTEREST = 31212534
    BEST_BID = 92.920000
    BEST_ASK = 92.950000
    LAST_TRADE = 93.000000
    OPEN = 93.090000
    VWAP = 93.307500
    LAST_ALL_SESSIONS = 93.020000
    BID_ALL_SESSION = 93.000000
    ASK_ALL_SESSION = 93.020000
    TRADING_DT_REALTIME = 2009-01-29
    EQY_TURNOVER_REALTIME = 1987223541.981339
    TOT_CALL_VOLUME_CUR_DAY_RT = 12824
    TOT_PUT_VOLUME_CUR_DAY_RT = 18332
    TOT_OPT_VOLUME_CUR_DAY_RT = 31156
    FUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
    IN_AUCTION_RT = false
    RT_API_MACHINE = n208
    ALL_PRICE_SIZE = 400
    ALL_PRICE = 93.020000
    ALL_PRICE_COND_CODE =
}```
LAST_AT_TRADE_TDY = 0.000000
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0.000000
HIGH_YLD_TDY = 0.000000
LOW_YLD_TDY = 0.000000
LAST_YLD_TDY = 0.000000
MID_TDY = 0.000000
SIZE_LAST_TRADE_TDY =
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 93.090000
ASK_SIZE_TDY = 1
BID_SIZE_TDY = 1
VOLUME_TDY = 21170839
LAST_PRICE_TDY = 93.000000
BID_TDY = 92.920000
ASK_TDY = 92.950000
HIGH_TDY = 94.340000
LOW_TDY = 92.600000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 93.070000
LAST_DIR = -1
LAST2_DIR = 1
RT_PRICING_SOURCE = US
SIZE_LAST_TRADE =
ASK_SIZE = 1
BID_SIZE = 1
API_MACHINE = n208
EXCH_CODE_LAST =
EXCH_CODE_BID = Q
EXCH_CODE_ASK = O
TRADE_SIZE_ALL_SESSIONS_RT = 400
IS_DELAYED_STREAM = false
EID = 14005
PREV_SES_LAST_PRICE = 94.200000
RT_PX_CHG_NET_1D = -1.200000
RT_PX_CHG_PCT_1D = -1.273890
TIME = 22:20:00.000+00:00
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
E.3 Asynchronous Event Handling

```cpp
// AsynchronousEventHandling.cpp

#include <blpapi_correlationid.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <iostream>
#include <string.h>  // for strcmp(3C)
#include <unistd.h>  // for pause(2)

using namespace BloombergLP;
using namespace blpapi;

namespace {

    // =========================
    // class RefDataEventHandler
    // =========================

    class RefDataEventHandler: public EventHandler
    {
        private:
            static void dumpEvent(const Event& event);
        public:
            // CREATORS
            RefDataEventHandler();
            ~RefDataEventHandler();

            // MANIPULATORS
            bool processEvent(const Event& event, Session *session);
        
            // CREATORS
            RefDataEventHandler::RefDataEventHandler()
            {
            }

            RefDataEventHandler::~RefDataEventHandler()
            {
            }
    }

    // =========================
    // class RefDataEventHandler
    // =========================

    namespace {

        // =========================
        // class RefDataEventHandler
        // =========================

        class RefDataEventHandler: public EventHandler
        {
            private:
                static void dumpEvent(const Event& event);
            public:
                // CREATORS
                RefDataEventHandler();
                ~RefDataEventHandler();

                // MANIPULATORS
                bool processEvent(const Event& event, Session *session);
            
                // CREATORS
                RefDataEventHandler::RefDataEventHandler()
                {
                }

                RefDataEventHandler::~RefDataEventHandler()
                {
                }
    }
```
bool RefDataEventHandler::processEvent(const Event& event, Session *session)
{
    switch (event.eventType()) {
    case Event::SESSION_STATUS: {
        MessageIterator iter(event);
        while (iter.next()) {
            Message message = iter.message();
            if (0 == ::strcmp("SessionStarted", message.messageType().string())) {
                session->openServiceAsync("//blp/refdata", CorrelationId((long long)99));
            } else {
                std::cerr << "Session Start Failure" << std::endl;
                message.print(std::cerr);
                ::exit(1);
            }
        }
        break;
    }
    case Event::SERVICE_STATUS: {
        MessageIterator iter(event);
        iter.next();
        Message message = iter.message();
        if (message.correlationId() == 99 && 0 == ::strcmp("ServiceOpened", message.messageType().string())) {
            // Construct and issue a Request
            Service service = session->getService("//blp/refdata");
            Request request = service.createRequest("ReferenceDataRequest");
            request.append("securities", "IBM US Equity");
            request.append("fields", "LAST_PRICE");
            session->sendRequest(request, CorrelationId((long long)86));
        } else {
            std::cerr << "Unexpected message" << std::endl;
            message.print(std::cerr);
            ::exit(1);
        }
        break;
    }
    case Event::PARTIAL_RESPONSE: {
        dumpEvent(event);
        break;
    }
    case Event::RESPONSE: {
        dumpEvent(event);
        session->stop();
        std::cout << "terminate process from handler" << std::endl;
        ::exit(0);
        break;
    }

default: {
    std::cerr << "Unexpected Event Type"
    << event.eventType()
    << std::endl;
    ::exit(1);
    break;
}

return true;

void RefDataEventHandler::dumpEvent(const Event& event)
{
    std::cout << "eventType="
    << event.eventType()
    << std::endl;
    MessageIterator messageIterator(event);
    while (messageIterator.next()) {
        Message message = messageIterator.message();
        std::cout << "messageType="
        << message.messageType()
        << std::endl;
        std::cout << "CorrelationId="
        << message.correlationId()
        << std::endl;
        message.print(std::cout);
    }
}

}  // close unnamed namespace

int main()
{
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);
    RefDataEventHandler refDataEventHandler;

    Session session(sessionOptions, &refDataEventHandler);
    // Start Session
    if (!session.startAsync()) {
        std::cerr << "Failed to start async session." << std::endl;
        return 1;
    }

    ::pause();

    return 0;
}
Asynchronous Event Handling: Output

```
eventType=5
messageType=ReferenceDataResponse
CorrelationId=[ valueType=INT classId=0 value=86 ]
ReferenceDataResponse = {
    securityData[] =
        securityData = {
            security = IBM US Equity
            eidData[] =
                fieldExceptions[] =
                    sequenceNumber = 0
                    fieldData = {
                        LAST_PRICE = 92.510000
                    }
        }
}
terminate process from handler
```
#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <iostream>
#include <string.h>  // for strcmp(3C)

using namespace BloombergLP;
using namespace blpapi;

static void handleResponseEvent(const Event& event)
{
    MessageIterator iter(event);
    while (iter.next()) {
        Message message = iter.message();
        Element referenceDataResponse = message.asElement();
        if (referenceDataResponse.hasElement("responseError")) {
            message.print(std::cout);
            ::exit(1);
        }

        Element securityDataArray =
            referenceDataResponse.getElement("securityData");
        int numItems = securityDataArray.numValues();

        for (int i = 0; i < numItems; ++i) {
            Element securityData = securityDataArray.getValueAsElement(i);
            std::string security =
                securityData.getElementAsString("security");
            int sequenceNumber =
                securityData.getElementAsInt32("sequenceNumber");
            if (securityData.hasElement("securityError")) {
                Element securityError =
                    securityData.getElement("securityError");
                std::cout << "* security =" << security << std::endl;
                securityError.print(std::cout);
            }
            return;
        }
    }
}
Element fieldData = securityData.getElement("fieldData");
double px_last = fieldData.getElementAsFloat64("PX_LAST");
std::string ds002 = fieldData.getElementAsString("DS002");
double vwap_volume = fieldData.getElementAsFloat64("VWAP_VOLUME");

// Individually output each value.
std::cout << "* security =" << security << "\n"
    << "* sequenceNumber=" << sequenceNumber << "\n"
    << "* px_last =" << px_last << "\n"
    << "* ds002 =" << ds002 << "\n"
    << "* vwap_volume =" << vwap_volume << "\n"
    << std::endl;
}

}
}

static void handleOtherEvent(const Event& event)
{
    std::cout << "EventType="
        << event.eventType()
        << std::endl;
    MessageIterator iter(event);
    while (iter.next()) {
        Message message = iter.message();
        std::cout << "correlationId="
            << message.correlationId()
            << std::endl;
        std::cout << "messageType="
            << message.messageType()
            << std::endl;
        message.print(std::cout);
        if (Event::SESSION_STATUS == event.eventType()
            && 0 == ::strcmp("SessionTerminated", message.messageType().string())){
            std::cout << "Terminating: "
                << message.messageType()
                << std::endl;
            ::exit(1);
        }
    }
}
}
int main()
{
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);

    Session session(sessionOptions); // Establish session
    // Start Session
    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return 1;
    }

    if (!session.openService("//blp/refdata")){
        std::cerr << "Failed to open service //blp/refdata." << std::endl;
        return 1;
    }

    CorrelationId requestId(1);
    Service refDataSvc = session.getService("//blp/refdata");

    Request request = refDataSvc.createRequest("ReferenceDataRequest");
    // append fields to request
    std::cout << "Initialize Request" << std::endl;
    request.getElement("securities").appendValue("AAPL US Equity");
    request.getElement("securities").appendValue("IBM US Equity");
    request.getElement("securities").appendValue("BLAHBLAHBLAH US Equity");
    request.getElement("fields").appendValue("PX_LAST");
    request.getElement("fields").appendValue("DS002");
    request.getElement("fields").appendValue("VWAP_VOLUME");
    // Volume used to calculate the Volume Weighted Average Price (VWAP)
    session.sendRequest(request, CorrelationId(1));

    bool continueToLoop = true;
    while (continueToLoop) {
        Event event = session.nextEvent();
        switch (event.eventType()) {
            case Event::RESPONSE: // final event
                continueToLoop = false; // fall through
                break;
            case Event::PARTIAL_RESPONSE:
                handleResponseEvent(event);
                break;
            default:
                handleOtherEvent(event);
                break;
        }
    }
}
```cpp
session.stop();
return 0;
}

Request Response Multiple: Output

Initialize Request
EventType=2
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=SessionStarted
SessionStarted = {
}
EventType=9
correlationId=[ valueType=UNSET classId=0 value=0 ]
messageType=ServiceOpened
ServiceOpened = {
    * security =AAPL US Equity
    * sequenceNumber=0
    * px_last =91.3
    * ds002 =APPLE INC
    * vwap_volume =1.31384e+07

    * security =IBM US Equity
    * sequenceNumber=1
    * px_last =92.37
    * ds002 =INTL BUSINESS MACHINES CORP
    * vwap_volume =4.22627e+06

    * security =BLAHLAHLAHL US Equity
securityError = {
    source = 119::bbdbs1
code = 15
category = BAD_SEC
message = Unknown/Invalid security [nid:119]
subcategory = INVALID_SECURITY
}
```
// SubscriptionMultiple.cpp

#include <blpapi_correlationid.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>
#include <blpapi_subscriptionlist.h>

#include <iostream>
#include <cassert>
#include <string>
#include <unistd.h>  // for pause(2)

using namespace BloombergLP;
using namespace blpapi;

namespace {
    // ==============================
    // class SubscriptionEventHandler
    // ==============================

class SubscriptionEventHandler: public EventHandler
{
    std::string     d_label;
    std::ostream   *d_stream; // held

    void            handleDataEvent  (const Event&   event,
                                        const Session& session);
    void            handleStatusEvent(const Event&   event,
                                       const Session& session);
    void            handleOtherEvent (const Event&   event,
                                       const Session& session);

    void            dumpEvent(const Event& event);

public:
    // CREATORS
    SubscriptionEventHandler(const std::string&  label,
                              std::ostream       *stream);
    ~SubscriptionEventHandler();

    // MANIPULATORS
    bool processEvent(const Event& event, Session *session);
};
// CREATORS
SubscriptionEventHandler::SubscriptionEventHandler(const std::string& label, std::ostream *stream)
    : d_label(label), d_stream(stream)
{
    assert(d_stream);
}

SubscriptionEventHandler::~SubscriptionEventHandler()
{
}

// MANIPULATORS
bool SubscriptionEventHandler::processEvent(const Event& event, Session *session)
{
    assert(session);
    switch (event.eventType()) {
        case Event::SUBSCRIPTION_DATA:
            handleDataEvent(event, *session);
            break;
        case Event::SESSION_STATUS:
        case Event::SERVICE_STATUS:
        case Event::SUBSCRIPTION_STATUS:
            handleStatusEvent(event, *session);
            break;
        default:
            handleOtherEvent(event, *session);
            break;
    }
    return true;
}

void SubscriptionEventHandler::dumpEvent(const Event& event)
{
    *d_stream << "handler label="
        << d_label
        << std::endl
        << "eventType="
        << event.eventType()
        << std::endl;
}
MessageIterator messageIterator(event);
while (messageIterator.next()) {
    Message message = messageIterator.message();
    *d_stream << "messageType=
        " << message.messageType() << std::endl
        "CorrelationId=
        " << message.correlationId() << std::endl;
    message.print(*d_stream);
}
}

void SubscriptionEventHandler::handleDataEvent(const Event& event,
        const Session& session)
{
    *d_stream << "handleDataEventHandler: enter" << std::endl;
    dumpEvent(event);
    *d_stream << "handleDataEventHandler: leave" << std::endl;
}

void SubscriptionEventHandler::handleStatusEvent(const Event& event,
        const Session& session)
{
    *d_stream << "handleStatusEventHandler: enter" << std::endl;
    dumpEvent(event);
    *d_stream << "handleStatusEventHandler: leave" << std::endl;
}

void SubscriptionEventHandler::handleOtherEvent(const Event& event,
        const Session& session)
{
    *d_stream << "handleOtherEvent: enter" << std::endl;
    dumpEvent(event);
    *d_stream << "handleOtherEvent: leave" << std::endl;
}
}  // close unnamed namespace
```cpp
int main(int argc, char **argv)
{
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("localhost");
    sessionOptions.setServerPort(8194);

    SubscriptionEventHandler
    subscriptionEventHandler(std::string("myLabel"), &std::cout);

    Session session(sessionOptions, &subscriptionEventHandler);

    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return 1;
    }

    if (!session.openService("//blp/mktdata")) {
        std::cerr << "Failed to open //blp/mktdata" << std::endl;
        return 1;
    }

    SubscriptionList subscriptions;
    subscriptions.add("IBM US Equity", "LAST_TRADE", "", CorrelationId((long long)10));
    subscriptions.add("/ticket/GOOG US Equity", "BID,ASK,LAST_PRICE", "", CorrelationId((long long)20));
    subscriptions.add("MSFTT US Equity", "LAST_PRICE", "interval=.5", CorrelationId((long long)30));
    subscriptions.add("/cusip/097023105?fields=LAST_PRICE&interval=5.0", "", "", CorrelationId((long long)40));
    session.subscribe(subscriptions);

    ::pause();

    return 0;
}
```
handleStatusEventHandler: enter
handler  label=myLabel
eventType=2
messageType=SessionStarted
CorrelationId= { valueType=UNSET classId=0 value=0 }
SessionStarted = {
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
handler  label=myLabel
eventType=9
messageType=ServiceOpened
CorrelationId= { valueType=UNSET classId=0 value=0 }
ServiceOpened = {
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
handler  label=myLabel
eventType=3
messageType=SubscriptionFailure
CorrelationId= { valueType=INT classId=0 value=30 }
SubscriptionFailure = {
    reason = {
        errorCode = 2
        description = Invalid security
category = BAD_SEC
        source = BBDB@n558
    }
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
handler  label=myLabel
eventType=3
messageType=SubscriptionStarted
CorrelationId= { valueType=INT classId=0 value=40 }
SubscriptionStarted = {
    exceptions[] =
}
messageType=SubscriptionStarted
CorrelationId= { valueType=INT classId=0 value=10 }
SubscriptionStarted = {
    exceptions[] =
}
messageType=SubscriptionStarted
CorrelationId= { valueType=INT classId=0 value=20 }
SubscriptionStarted = {
    exceptions[] =
}
handleStatusEventHandler: leave
handleDataEventHandler: enter
handler label=myLabel
eventType=8
messageType=MarketDataEvents
CorrelationId=[ valueType=INT classId=0 value=20 ]
MarketDataEvents = {
    LAST_PRICE = 338.460000
    BID = 338.360000
    ASK = 338.500000
    VOLUME = 4068281
    HIGH = 348.800000
    LOW = 336.001000
    BEST_BID = 338.360000
    BEST_ASK = 338.500000
    LAST_TRADE = 338.460000
    OPEN = 344.690000
    INDICATIVE_FAR = 344.690000
    INDICATIVE_NEAR = 344.690000
    IMBALANCE_BID =
    IMBALANCE_ASK = 344.760000
    VWAP = 341.666700
    LAST_ALL_SESSIONS = 338.460000
    IMBALANCE_INDIC_RT = SELL
    PREV_CLOSE_VALUE_REALTIME = 343.320000
    BID_ALL_SESSION = 338.360000
    ASK_ALL_SESSION = 338.500000
    TRADING_DT_REALTIME = 2009-01-30
    EQY_TURNOVER_REALTIME = 1379007507.741211
    TOT_CALL_VOLUME_CUR_DAY_RT = 3266
    TOT_PUT_VOLUME_CUR_DAY_RT = 4650
    TOT_OPT_VOLUME_CUR_DAY_RT = 7916
    PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 1
    IN_AUCTION_RT = false
    RT_API_MACHINE = p060
    ALL_PRICE_SIZE = 100
    ALL_PRICE = 338.460000
    ALL_PRICE_COND_CODE =
    BID_COND_CODE =
    ASK_COND_CODE =
    LAST_AT_TRADE_TDY = 0.000000
    SIZE_LAST_AT_TRADE_TDY = 0
    OPEN_YLD_TDY = 0.000000
    HIGH_YLD_TDY = 0.000000
    LOW_YLD_TDY = 0.000000
    LAST_YLD_TDY = 0.000000
    MID_TDY = 0.000000
    SIZE_LAST_TRADE_TDY = 100
    IND_BID_FLAG = false
    IND_ASK_FLAG = false
    OPEN_TDY = 344.690000
```
ASK_SIZE_TDY = 2
BID_SIZE_TDY = 3
VOLUME_TDY = 4068281
LAST_PRICE_TDY = 338.460000
BID_TDY = 338.360000
ASK_TDY = 338.500000
HIGH_TDY = 348.800000
LOW_TDY = 336.001000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 338.450000
LAST2_DIR = 1
BID2 = 338.360000
ASK2 = 338.500000
SIZE_LAST_TRADE = 100
ASK_SIZE = 2
BID_SIZE = 3
API_MACHINE = p060
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 100
IS_DELAYED_STREAM = false
EID = 14005
PREV SES LAST PRICE = 343.320000
RT_PX_CHG_NET_1D = -4.860000
RT_PX_CHG_PCT_1D = -1.415590
TIME = 20:48:30.000+00:00
LAST UPDATE_BID RT = 20:48:33.000+00:00
LAST UPDATE_ASK RT = 20:48:32.000+00:00
BID ASK TIME = 20:48:33.000+00:00
SES_START = 14:30:00.000+00:00
SES END = 21:30:00.000+00:00
```

```cpp
handleDataEventHandler: leave
handleDataEventHandler: enter
handler label=myLabel
eventType=8
messageType=MarketDataEvents
CorrelationId=[ valueType=INT classId=0 value=10 ]
MarketDataEvents = {
  LAST_PRICE = 91.830000
  BID = 91.820000
  ASK = 91.830000
  VOLUME = 7233307
  HIGH = 93.480000
  LOW = 91.250000
  BEST_BID = 91.820000
  BEST_ASK = 91.830000
  LAST_TRADE = 91.830000
```
OPEN = 92.230000
IMBALANCE_BID =
IMBALANCE_ASK = 91.780000
ORDER_IMB_BUY_VOLUME =
ORDER_IMB_SELL_VOLUME = 54500.000000
VWAP = 92.495700
THEO_PRICE = 0.000000
LAST_ALL_SESSIONS = 91.830000
IMBALANCE_INDIC_RT = SELL
PREV_CLOSE_VALUE_REALTIME = 92.510000
BID_ALL_SESSION = 91.820000
ASK_ALL_SESSION = 91.830000
TRADING_DT_REALTIME = 2009-01-30
EQY_TURNOVER_REALTIME = 666435537.542725
FINANCIAL_STATUS_INDICATOR_RT = 0
NYSE_LRP_HIGH_PRICE_RT = 92.850000
NYSE_LRP_LOW_PRICE_RT = 90.850000
TOT_CALL_VOLUME_CUR_DAY_RT = 2345
TOT_PUT_VOLUME_CUR_DAY_RT = 2282
TOT_OPT_VOLUME_CUR_DAY_RT = 4627
PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
IN_AUCTION_RT = false
RT_API_MACHINE = n160
ALL_PRICE_SIZE = 100
ALL_PRICE = 91.830000
ALL_PRICE_COND_CODE =
BID_COND_CODE =
ASK_COND_CODE =
VOLUME_THEO = 0
LAST_AT_TRADE_TDY = 0.000000
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0.000000
HIGH_YLD_TDY = 0.000000
LOW_YLD_TDY = 0.000000
LAST_YLD_TDY = 0.000000
MID_TDY = 0.000000
SIZE_LAST_TRADE_TDY = 100
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 92.230000
ASK_SIZE_TDY = 1
BID_SIZE_TDY = 2
VOLUME_TDY = 7233307
LAST_PRICE_TDY = 91.830000
BID_TDY = 91.820000
ASK_TDY = 91.830000
HIGH_TDY = 93.480000
LOW_TDY = 91.250000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 91.839000
LAST_DIR = -1
LAST2_DIR = 1
BID_DIR = -1
ASK_DIR = -1
BID2 = 91.820000
ASK2 = 91.830000
SIZE_LAST_TRADE = 100
ASK_SIZE = 1
BID_SIZE = 2
API_MACHINE = n160
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 100
IS_DELAYED_STREAM = false
EID = 14003
PREV_SES_LAST_PRICE = 92.510000
RT_PX_CHG_NET_1D = -0.679900
RT_PX_CHG_PCT_1D = -0.734947
TIME = 20:48:34.000+00:00
LAST_UPDATE_BID_RT = 20:48:34.000+00:00
LAST_UPDATE_ASK_RT = 20:48:34.000+00:00
NYSE_LRP_SEND_TIME_RT = 20:48:34.000+00:00
BID_ASK_TIME = 20:48:34.000+00:00
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
}
F  C Examples

This section contains the following code examples:

- “RequestResponseParadigm” on page 313
- “Subscription Paradigm” on page 318
- “Asynchronous Event Handling” on page 327
- “Request Response Multiple” on page 332
- “Subscription Multiple” on page 340

**Note:** These examples use `assert` statements to make manifest the program state at various key points. Follow your organization’s guidelines for best practices on the use of `assert` statements in production code.

**Note:** When using the C language interface the programmer must explicitly recover allocated resources such as sessions, session options, requests, and message iterators. In general, a pointer to a resource obtained from a function containing the word “create” must be recovered by invoking a similarly named function containing the word “destroy”. For example, the `blpapi_Service_createRequest` function delivers a pointer to a `blpapi_Request_t` type and that pointer, when no longer needed, must be passed to the `blpapi_Request_destroy` function.
/* RequestResponseParadigm.c */

#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h> /* for strcmp(3C) and memset(3C) */

static int streamWriter(const char* data, int length, void *stream)
{
    assert(data);
    assert(stream);
    return fwrite(data, length, 1, (FILE *)stream);
}

static void handleResponseEvent(const blpapi_Event_t *event)
{
    blpapi_MessageIterator_t *iter = 0;
    blpapi_Message_t *message = 0;
    assert(event);
    printf("Event Type = %d\n", blpapi_Event_eventType(event));
    iter = blpapi_MessageIterator_create(event);
    assert(iter);
    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_CorrelationId_t correlationId;
        blpapi_Element_t *messageElements = 0;
        assert(message);
        correlationId = blpapi_Message_correlationId(message, 0);
        printf("correlationId=%d %d %lld\n",
               correlationId.valueType,
               correlationId.classId,
               correlationId.value.intValue);
printf("messageType =\%s\n", blpapi_Message_typeString(message));
messageElements = blpapi_Message_elements(message);
assert(messageElements);
blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
}
blpapi_MessageIterator_destroy(iter);
}

static void handleOtherEvent(const blpapi_Event_t *event)
{
    blpapi_MessageIterator_t *iter = 0;
    blpapi_Message_t *message = 0;

    assert(event);
    printf("EventType=%d\n", blpapi_Event_eventType(event));
    iter = blpapi_MessageIterator_create(event);
    assert(iter);
    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_CorrelationId_t correlationId;
        blpapi_Element_t *messageElements = 0;

        assert(message);
        correlationId = blpapi_Message_correlationId(message, 0);
        printf("correlationId=%d %d %lld\n",
               correlationId.valueType,
               correlationId.classId,
               correlationId.value.intValue);

        printf("messageType=%s\n", blpapi_Message_typeString(message));
        messageElements = blpapi_Message_elements(message);
        assert(messageElements);
        blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);

        if (BLPAPI_EVENTTYPE_SESSION_STATUS ==
            blpapi_Event_eventType(event)
            && 0 == strcmp("SessionTerminated",
                           blpapi_Message_typeString(message))) {
            fprintf(stdout,
                    "Terminating: %s\n",
                    blpapi_Message_typeString(message));
            exit(1);
        }
    }
    blpapi_MessageIterator_destroy(iter);
}
int main()
{
    blpapi_SessionOptions_t *sessionOptions = 0;
    blpapi_Session_t *session = 0;
    blpapi_CorrelationId_t requestId;
    blpapi_Service_t *refDataSvc = 0;
    blpapi_Request_t *request = 0;
    blpapi_Element_t *elements = 0;
    blpapi_Element_t *securitiesElements = 0;
    blpapi_Element_t *fieldsElements = 0;
    int continueToLoop = 1;
    blpapi_CorrelationId_t correlationId;

    sessionOptions = blpapi_SessionOptions_create();
    assert(sessionOptions);

    blpapi_SessionOptions_setServerHost(sessionOptions, "localhost");
    blpapi_SessionOptions_setServerPort(sessionOptions, "8194");

    session = blpapi_Session_create(sessionOptions, 0, 0, 0);
    assert(session);

    blpapi_SessionOptions_destroy(sessionOptions);

    if (0 != blpapi_Session_start(session)) {
        fprintf(stderr, "Failed to start session.\n");
        blpapi_Session_destroy(session);
        return 1;
    }

    if (0 != blpapi_Session_openService(session, "//blp/refdata"){
        fprintf(stderr, "Failed to open service //blp/refdata.\n");
        blpapi_Session_destroy(session);
        return 1;
    }

    memset(&requestId, '\0', sizeof(requestId));
    requestId.size = sizeof(requestId);
    requestId.valueType = BLPAPI_CORRELATION_TYPE_INT;
    requestId.value.intValue = (blpapi_UInt64_t)1;

    blpapi_Session_getService(session, &refDataSvc, "//blp/refdata");

    blpapi_Service_createRequest(refDataSvc, &request,
        "ReferenceDataRequest");
    assert(request);
elements = blpapi_Request_elements(request);
assert(elements);

blpapi_Element_getElement(elements,
   &securitiesElements,
   "securities",
   0);
assert(securitiesElements);
blpapi_Element_setValueString(securitiesElements,
   "IBM US Equity",
   BLPAPI_ELEMENT_INDEX_END);
blpapi_Element_getElement(elements, &fieldsElements, "fields", 0);
blpapi_Element_setValueString(fieldsElements,
   "PX_LAST",
   BLPAPI_ELEMENT_INDEX_END);

memset(&correlationId, '\0', sizeof(correlationId));
correlationId.size     = sizeof(correlationId);
correlationId.valueType = BLPAPI_CORRELATION_TYPE_INT;
correlationId.value.intValue = (blpapi_UInt64_t)1;

blpapi_Session_sendRequest(session, request, &correlationId, 0, 0, 0,
                          0);

while (continueToLoop) {
   blpapi_Event_t *event = 0;
   blpapi_Session_nextEvent(session, &event, 0);
   assert(event);
   switch (blpapi_Event_eventType(event)) {
   case BLPAPI_EVENTTYPE_RESPONSE: // final event
      continueToLoop = 0;            // fall through
   case BLPAPI_EVENTTYPE_PARTIAL_RESPONSE:
      handleResponseEvent(event);
      break;
   default:
      handleOtherEvent(event);
      break;
   }
   blpapi_Event_release(event);
}

blpapi_Session_stop(session);

blpapi_Request_destroy(request);
blpapi_Session_destroy(session);

return 0;
Request Response Paradigm Output

```
EventType=2
  correlationId=0 0 0
messageType=SessionStarted
SessionStarted = {
}
EventType=9
  correlationId=0 0 0
messageType=ServiceOpened
ServiceOpened = {
}
EventType=5
  correlationId=1 0 1
messageType=ReferenceDataResponse
ReferenceDataResponse = {
  securityData[] =
    securityData = {
      security = IBM US Equity
      eidData[] =
        fieldExceptions[] =
          sequenceNumber = 0
          fieldData = {
            PX_LAST = 91.170000
          }
    }
}
```
/ * SubscriptionParadigm.c */

#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>
#include <blpapi_subscriptionlist.h>

#include <assert.h>
#include <stdio.h>
#include <stdlib.h> /* for exit(2) */
#include <string.h> /* for strcmp(3C) and memset(3C) */

static int streamWriter(const char* data, int length, void *stream)
{
    assert(data);
    assert(stream);
    return fwrite(data, length, 1, (FILE *)stream);
}

static void handleDataEvent(const blpapi_Event_t *event, int updateCount)
{
    blpapi_MessageIterator_t *iter = 0;
    blpapi_Message_t *message = 0;

    assert(event);

    printf("EventType=%d\n", blpapi_Event_eventType(event));
    printf("updateCount = %d\n", updateCount);

    iter = blpapi_MessageIterator_create(event);
    assert(iter);

    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_CorrelationId_t correlationId;
        blpapi_Element_t *messageElements = 0;

        assert(message);

        correlationId = blpapi_Message_correlationId(message, 0);
        printf("correlationId=%d %d %lld\n",
                correlationId.valueType,
                correlationId.classId,
                correlationId.value.intValue);
    }
}
printf("messageType = %s\n", blpapi_Message_typeString(message));
messageElements = blpapi_Message_elements(message);
blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
}
blpapi_MessageIterator_destroy(iter);
}

static void handleOtherEvent(const blpapi_Event_t *event)
{
  blpapi_MessageIterator_t *iter    = 0;
  blpapi_Message_t         *message = 0;

  assert(event);
  printf("EventType=%d\n", blpapi_Event_eventType(event));
  iter = blpapi_MessageIterator_create(event);
  assert(iter);

  while (0 == blpapi_MessageIterator_next(iter, &message)) {
    blpapi_CorrelationId_t  correlationId;
    blpapi_Element_t       *messageElements = 0;

    assert(message);

    correlationId = blpapi_Message_correlationId(message, 0);
    printf("correlationId=%d %d %lld\n",
            correlationId.valueType,
            correlationId.classId,
            correlationId.value.intValue);

    printf("messageType=%s\n", blpapi_Message_typeString(message));
    messageElements = blpapi_Message_elements(message);
    blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);

    if (BLPAPI_EVENTTYPE_SESSION_STATUS ==
        blpapi_Event_eventType(event)
        && 0 == strcmp("SessionTerminated",
                      blpapi_Message_typeString(message))){
      fprintf(stdout,
              "Terminating: %s\n",
              blpapi_Message_typeString(message));
      exit(1);
    }
  }

  blpapi_MessageIterator_destroy(iter);
}
int main()
{
    blpapi_SessionOptions_t *sessionOptions = 0;
    blpapi_Session_t *session = 0;
    blpapi_CorrelationId_t subscriptionId;
    blpapi_SubscriptionList *subscriptions = 0;
    const char *fields[1] = {"LAST_PRICE");
    const char **options = 0;
    int updateCount = 0;

    setbuf(stdout, 0); /* NO SHOW */

    sessionOptions = blpapi_SessionOptions_create();
    assert(sessionOptions);
    blpapi_SessionOptions_setServerHost(sessionOptions, "localhost");
    blpapi_SessionOptions_setServerPort(sessionOptions, "8194");

    session = blpapi_Session_create(sessionOptions, 0, 0, 0);
    assert(session);

    blpapi_SessionOptions_destroy(sessionOptions);

    if (0 != blpapi_Session_start(session)) {
        fprintf(stderr, "Failed to start session.\n");
        blpapi_Session_destroy(session);
        return 1;
    }

    if (0 != blpapi_Session_openService(session, "/blp/mktdata")){
        fprintf(stderr, "Failed to open service \
blpapi_Session_destroy(session);
        return 1;
    }

    memset(&subscriptionId, '\0', sizeof(subscriptionId));
    subscriptionId.size = sizeof(subscriptionId);
    subscriptionId.valueType = BLPAPI_CORRELATION_TYPE_INT;
    subscriptionId.value.intValue = (blpapi_UInt64_t)2;

    subscriptions = blpapi_SubscriptionList_create();
    assert(subscriptions);
blpapi_SubscriptionList_add(subscriptions,
    "AAPL US Equity",
    &subscriptionId,
    fields,
    options,
    1,
    0);

blpapi_Session_subscribe(session,
    subscriptions,
    0,
    0,
    0);

while (1) {
    blpapi_Event_t *event = 0;
    blpapi_Session_nextEvent(session, &event, 0);
    assert(event);

    switch (blpapi_Event_eventType(event)) {
    case BLPAPI_EVENTTYPE_SUBSCRIPTION_DATA:
        handleDataEvent(event, updateCount++);
        break;
    default:
        handleOtherEvent(event);
        break;
    }
    blpapi_Event_release(event);
}

return 0;
EventType=2
correlationId=0 0 0
messageType=SessionStarted
SessionStarted = {
}

EventType=9
correlationId=0 0 0
messageType=ServiceOpened
ServiceOpened = {
}

EventType=3
correlationId=1 0 2
messageType=SubscriptionStarted
SubscriptionStarted = {
  exceptions[] =
}

EventType=8
updateCount = 0
correlationId=1 0 2
messageType = MarketDataEvents
MarketDataEvents = {
  LAST_PRICE = 90.886000
  BID = 90.880000
  ASK = 90.910000
  VOLUME = 7596090
  HIGH = 91.640000
  LOW = 88.900000
  BEST_BID = 90.880000
  BEST_ASK = 90.910000
  LAST_TRADE = 90.886000
  OPEN = 89.100000
  INDICATIVE_FAR = 89.130000
  INDICATIVE_NEAR = 89.130000
  IMBALANCE_BID =
  IMBALANCE_ASK =
  VWAP = 90.159300
  LAST_ALL_SESSIONS = 90.886000
  IMBALANCE_INDIC_RT = NOIM
  BID_ALL_SESSION = 90.880000
  ASK_ALL_SESSION = 90.910000
  TRADING_DT_REALTIME = 2009-02-02
  EQY_TURNOVER_REALTIME = 682873786.088959
  TOT_CALL_VOLUME_CUR_DAY_RT = 4886
  TOT_PUT_VOLUME_CUR_DAY_RT = 3457
  TOT_OPT_VOLUME_CUR_DAY_RT = 8343
  PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
  IN_AUCTION_RT = false
  RT_API_MACHINE = n125
  ALL_PRICE_SIZE = 1000
ALL_PRICE = 90.886000
ALL_PRICE_COND_CODE =
BID_COND_CODE =
ASK_COND_CODE =
LAST_AT_TRADE_TDY = 0.000000
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0.000000
HIGH_YLD_TDY = 0.000000
LOW_YLD_TDY = 0.000000
LAST_YLD_TDY = 0.000000
MID_TDY = 0.000000
SIZE_LAST_TRADE_TDY = 1000
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 89.100000
ASK_SIZE_TDY = 5
BID_SIZE_TDY = 7
VOLUME_TDY = 7596090
LAST_PRICE_TDY = 90.886000
BID_TDY = 90.880000
ASK_TDY = 90.910000
HIGH_TDY = 91.640000
LOW_TDY = 88.900000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 90.900000
LAST_DIR = -1
LAST2_DIR = 1
BID_DIR = 1
ASK_DIR = 1
BID2 = 90.880000
ASK2 = 90.910000
SIZE_LAST_TRADE = 1000
ASK_SIZE = 5
BID_SIZE = 7
API_MACHINE = n166
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 1000
IS_DELAYED_STREAM = false
EID = 14005
PREV_SES_LAST_PRICE = 90.130000
RT_PX_CHG_NET_1D = 0.756000
RT_PX_CHG_PCT_1D = 0.838788
TIME = 16:36:33.000+00:00
LAST_UPDATE_BID_RT = 16:36:35.000+00:00
LAST_UPDATE_ASK_RT = 16:36:32.000+00:00
BID_ASK_TIME = 16:36:35.000+00:00
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
}
EventType=8
updateCount = 1
correlationId=1 0 2
messageType = MarketDataEvents
MarketDataEvents = {
LAST_PRICE = 90.886000
BID = 90.880000
ASK = 90.910000
VOLUME = 7596090
HIGH = 91.640000
LOW = 88.900000
BEST_BID = 90.880000
BEST_ASK = 90.910000
LAST_TRADE = 90.886000
VWAP = 90.644800
LAST_ALL_SESSIONS = 90.886000
BID_ALL_SESSION = 90.880000
ASK_ALL_SESSION = 90.910000
EQY_TURNOVER_REALTIME = 682873786.088959
TOT_CALL_VOLUME_CUR_DAY_RT = 4886
TOT_PUT_VOLUME_CUR_DAY_RT = 3457
TOT_OPT_VOLUME_CUR_DAY_RT = 8343
PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
IN_AUCTION_RT = false
ALL_PRICE_SIZE = 1000
ALL_PRICE = 90.886000
ALL_PRICE_COND_CODE =
LAST_AT_TRADE_TDY = 0.000000
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0.000000
HIGH_YLD_TDY = 0.000000
LOW_YLD_TDY = 0.000000
LAST_YLD_TDY = 0.000000
MID_TDY = 0.000000
SIZE_LAST_TRADE_TDY = 1000
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 89.100000
ASK_SIZE_TDY = 5
BID_SIZE_TDY = 7
VOLUME_TDY = 7596090
LAST_PRICE_TDY = 90.886000
BID_TDY = 90.880000
ASK_TDY = 90.910000
HIGH_TDY = 91.640000
LOW_TDY = 88.900000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 90.900000
LAST_DIR = -1
LAST2_DIR = 1
BID_DIR = 1
ASK_DIR = 1
BID2 = 90.880000
ASK2 = 90.910000
SIZE_LAST_TRADE = 1000
ASK_SIZE = 5
BID_SIZE = 7
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 1000
IS_DELAYED_STREAM = false
EID = 14005
RT_PX_CHG_NET_1D = 0.756000
RT_PX_CHG_PCT_1D = 0.838788
TIME = 16:36:33.000+00:00
LAST_UPDATE_BID_RT = 16:36:35.000+00:00
LAST_UPDATE_ASK_RT = 16:36:32.000+00:00
BID_ASK_TIME = 16:36:35.000+00:00
}
EventType=8
updateCount = 2
correlationId=1 0 2
messageType = MarketDataEvents
MarketDataEvents = {
    LAST2_PRICE = 90.886000
    LAST_PRICE = 90.910000
    LAST_ALL_SESSIONS = 90.910000
    LAST_PRICE_TDY = 90.910000
    LAST2_DIR = -1
    LAST_DIR = 1
EQY_TURNOVER_REALTIME = 682882877.088959
SIZE_LAST_TRADE = 100
SIZE_LAST_TRADE_TDY = 100
TRADE_SIZE_ALL_SESSIONS_RT = 100
VOLUME = 7596190
VOLUME_TDY = 7596190
LAST_TRADE = 90.910000
ALL_PRICE = 90.910000
ALL_PRICE_SIZE = 100
EID = 14005
RT_PX_CHG_NET_1D = 0.780000
RT_PX_CHG_PCT_1D = 0.865417
IS_DELAYED_STREAM = false
TIME = 16:36:37.000+00:00
EVENT_TIME = 16:36:37.000+00:00
}
EventType=8
correlationId=1 0 2
messageType = MarketDataEvents
MarketDataEvents = {
    LAST2_PRICE = 90.910000
    LAST_PRICE = 90.910000
    LAST_ALL_SESSIONS = 90.910000
    LAST_PRICE_TDY = 90.910000
    LAST2_DIR = 1
    EQY_TURNOVER_REALTIME = 682891968.088959
    SIZE_LAST_TRADE = 100
    SIZE_LAST_TRADE_TDY = 100
    TRADE_SIZE_ALL_SESSIONS_RT = 100
    VOLUME = 7596290
    VOLUME_TDY = 7596290
    LAST_TRADE = 90.910000
    ALL_PRICE = 90.910000
    ALL_PRICE_SIZE = 100
    EID = 14005
    RT_PX_CHG_NET_1D = 0.780000
    RT_PX_CHG_PCT_1D = 0.865417
    IS_DELAYED_STREAM = false
    TIME = 16:36:37.000+00:00
    EVENT_TIME = 16:36:37.000+00:00
}
correlationId=1 0 2
messageType = MarketDataEvents
MarketDataEvents = {
    LAST2_PRICE = 90.910000
    LAST_PRICE = 90.910000
    LAST_ALL_SESSIONS = 90.910000
    LAST_PRICE_TDY = 90.910000
    LAST2_DIR = 1
    EQY_TURNOVER_REALTIME = 682901059.088959
    SIZE_LAST_TRADE = 100
    SIZE_LAST_TRADE_TDY = 100
    TRADE_SIZE_ALL_SESSIONS_RT = 100
    VOLUME = 7596390
    VOLUME_TDY = 7596390
    LAST_TRADE = 90.910000
    ALL_PRICE = 90.910000
    ALL_PRICE_SIZE = 100
    EID = 14005
    RT_PX_CHG_NET_1D = 0.780000
    RT_PX_CHG_PCT_1D = 0.865417
    IS_DELAYED_STREAM = false
    TIME = 16:36:37.000+00:00
    EVENT_TIME = 16:36:37.000+00:00
}
/* RequestResponseParadigm.c */
#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>  /* for exit(2) */
#include <string.h>  /* for strcmp(3C) and memset(3C) */
#include <unistd.h>  /* for pause(2) */

static int streamWriter(const char* data, int length, void *stream)
{
    assert(data);
    assert(stream);
    return fwrite(data, length, 1, (FILE *)stream);
}

static void dumpEvent(blpapi_Event_t *event) /* not const! */
{
    blpapi_MessageIterator_t *iter    = 0;
    blpapi_Message_t         *message = 0;
    assert(event);
    printf("eventType=%d\n", blpapi_Event_eventType(event));
    iter = blpapi_MessageIterator_create(event);
    assert(iter);
    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_CorrelationId_t  correlationId;
        blpapi_Element_t       *messageElements = 0;
        assert(message);
        printf("messageType=%s\n", blpapi_Message_typeString(message));
        correlationId = blpapi_Message_correlationId(message, 0);
        printf("correlationId=%d %d %lld\n",
            correlationId.valueType,
            correlationId.classId,
            correlationId.intValue);
messageElements = blpapi_Message_elements(message);
assert(messageElements);
blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
}
}
#endif
extern "C"
#endif
static void processEvent(blpapi_Event_t *event,
blpapi_Session_t *session,
void *userData)
{
assert(event);
assert(session);

switch (blpapi_Event_eventType(event)) {
    case BLPAPI_EVENTTYPE_SESSION_STATUS: {
        blpapi_MessageIterator_t *iter = 0;
        blpapi_Message_t *message = 0;

        iter = blpapi_MessageIterator_create(event);
        assert(iter);
        while (0 == blpapi_MessageIterator_next(iter, &message)) {
            if (0 == strcmp("SessionStarted",
                            blpapi_Message_typeString(message))) {
                blpapi_CorrelationId_t correlationId;
                memset(&correlationId, '\0', sizeof(correlationId));
                correlationId.size       = sizeof(correlationId);
                correlationId.valueType  = BLPAPI_CORRELATION_TYPE_INT;
                correlationId.value.intValue = (blpapi_UInt64_t)99;

                blpapi_Session_openServiceAsync(session,
                                                "/blp/refdata",
                                                &correlationId);
            } else {
                blpapi_Element_t *messageElements = 0;

                messageElements = blpapi_Message_elements(message);
                assert(messageElements);
                blpapi_Element_print(messageElements,
                                      &streamWriter,
                                      stdout,
                                      0,
                                      4);
            }
        }
    }
    break;
case BLPAPI_EVENTTYPE_SERVICE_STATUS: {
    blpapi_MessageIterator_t *iter       = 0;
    blpapi_Message_t         *message    = 0;
    blpapi_Service_t         *refDataSvc = 0;
    blpapi_CorrelationId_t    correlationId;

    iter = blpapi_MessageIterator_create(event);
    assert(iter);

    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        assert(message);
        correlationId = blpapi_Message_correlationId(message, 0);

        if (correlationId.value.intValue == (blpapi_UInt64_t)99
            &&  0 == strcmp("ServiceOpened",
              blpapi_Message_typeString(message))) {
            blpapi_Request_t *request            = 0;
            blpapi_Element_t *elements           = 0;
            blpapi_Element_t *securitiesElements = 0;
            blpapi_Element_t *fieldsElements     = 0;

            /* Construct and issue a Request */
            blpapi_Session_getService(session,
                &refDataSvc,
                "/blp/refdata");

            blpapi_Service_createRequest(refDataSvc,
                &request,
                "ReferenceDataRequest");
            assert(request);
            elements = blpapi_Request_elements(request);
            assert(elements);

            blpapi_Element_getElement(elements,
                &securitiesElements,
                "securities",
                0);
            assert(securitiesElements);
            blpapi_Element_setValueString(securitiesElements,
                "IBM US Equity",
                BLPAPI_ELEMENT_INDEX_END);

            blpapi_Element_getElement(elements,
                &fieldsElements,
                "fields",
                0);
            blpapi_Element_setValueString(fieldsElements,
                "PX_LAST",
                BLPAPI_ELEMENT_INDEX_END);
memset(&correlationId, '\0', sizeof(correlationId));
correlationId.size = sizeof(correlationId);
correlationId.valueType = BLPAPI_CORRELATION_TYPE_INT;
correlationId.value.intValue = (blpapi_UInt64_t)86;

blpapi_Session_sendRequest(session,
                          request,
                          &correlationId,
                          0,
                          0,
                          0,
                          0);
}

else {
    blpapi_Element_t *messageElements = 0;
    fprintf(stderr, "Unexpected message\n");

    messageElements = blpapi_Message_elements(message);
    assert(messageElements);
    blpapi_Element_print(messageElements,
                          &streamWriter,
                          stdout,
                          0,
                          4);
}
break;
}

break;

}

break;

}

}

default: {
    fprintf(stderr, "default-case\n");
    fprintf(stderr, "Unexpected Event Type %d\n",
             blpapi_Event_eventType(event));
    exit(1);
    break;
}
break;
}

}
int main()
{
    blpapi_SessionOptions_t *sessionOptions = 0;
    blpapi_Session_t        *session       = 0;

    sessionOptions = blpapi_SessionOptions_create();
    assert(sessionOptions);

    blpapi_SessionOptions_setServerHost(sessionOptions, "localhost");
    blpapi_SessionOptions_setServerPort(sessionOptions, "8194");

    session = blpapi_Session_create(sessionOptions, &processEvent, 0, 0);
    assert(session);

    blpapi_SessionOptions_destroy(sessionOptions);

    if (0 != blpapi_Session_start(session)) {
        fprintf(stderr, "Failed to start async session.\n");
        blpapi_Session_destroy(session);
        return 1;
    }

    pause();

    blpapi_Session_destroy(session);
    return 0;
}

Asynchronous Event Handling Output

eventType=5
messageType=ReferenceDataResponse
correlationId=1 0 86
ReferenceDataResponse = {
    securityData[] =
        securityData = {
            security = IBM US Equity
            eidData[] =
                fieldExceptions[] =
                    sequenceNumber = 0
                fieldData = {
                    PX_LAST = 91.170000
                }
        }
}

terminate process from handler
/* RequestResponseParadigm.c */

#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>

#include <assert.h>
#include <stdio.h>
#include <string.h>  /* for strcmp(3C) */

static int streamWriter(const char* data, int length, void *stream)
{
    assert(data);
    assert(stream);
    return fwrite(data, length, 1, (FILE *)stream);
}

static void handleResponseEvent(const blpapi_Event_t *event)
{
    blpapi_MessageIterator_t *iter    = 0;
    blpapi_Message_t         *message = 0;

    assert(event);
    iter = blpapi_MessageIterator_create(event);
    assert(iter);

    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_Element_t       *referenceDataResponse = 0;
        blpapi_Element_t       *securityDataArray     = 0;
        int                     numItems              = 0;

        assert(message);

        referenceDataResponse = blpapi_Message_elements(message);
        assert(referenceDataResponse);

        if (blpapi_Element_hasElement(referenceDataResponse,
                                        "responseError", 0)) {
            // Further processing...
        }
fprintf(stderr, "has responseError\n");
blpapi_Element_print(referenceDataResponse, &streamWriter, stdout, 0, 4);
exit(1);
}

blpapi_Element_getElement(referenceDataResponse, &securityDataArray, "securityData",
0);
numItems = blpapi_Element_numValues(securityDataArray);

for (int i = 0; i < numItems; ++i) {
    blpapi_Element_t *securityData = 0;
    blpapi_Element_t *securityElement = 0;
    const char *security = 0;
    blpapi_Element_t *sequenceNumberElement = 0;
    int sequenceNumber = -1;

    blpapi_Element_getValueAsElement(securityDataArray, &securityData, i);
    assert(securityData);

    blpapi_Element_getElement(securityData, &securityElement, "security",
0);
    assert(securityElement);
    blpapi_Element_getValueAsString(securityElement, &security, 0);
    assert(security);

    blpapi_Element_getElement(securityData, &sequenceNumberElement, "sequenceNumber",
0);
    assert(sequenceNumberElement);
    blpapi_Element_getValueAsInt32(sequenceNumberElement, &sequenceNumber, 0);
if (blpapi_Element_hasElement(securityData, "securityError", 0)) {
    blpapi_Element_t *securityErrorElement = 0;
    printf("*security =\n", security);
    blpapi_Element_getElement(securityData,
        &securityErrorElement, "securityError", 0);
    assert(securityErrorElement);
    blpapi_Element_print(securityErrorElement, &streamWriter, stdout, 0, 4);
    return;
} else {
    blpapi_Element_t *fieldDataElement = 0;
    blpapi_Element_t *PX_LAST_Element = 0;
    blpapi_Element_t *DS002_Element = 0;
    blpapi_Element_t *VWAP_VOLUME_Element = 0;
    double px_last = (double)777;
    const char *ds002 = 0;
    double vwap_volume = (double)666;

    blpapi_Element_getElement(securityData, &fieldDataElement, "fieldData", 0);
    assert(fieldDataElement);
    blpapi_Element_getElement(fieldDataElement, &PX_LAST_Element, "PX_LAST", 0);
    assert(PX_LAST_Element);
    blpapi_Element_getValueAsFloat64(PX_LAST_Element, &px_last, 0);
    blpapi_Element_getElement(fieldDataElement, &DS002_Element, "DS002", 0);
    assert(DS002_Element);
    blpapi_Element_getValueAsString(DS002_Element, &ds002, 0);
blpapi_Element_getElement(fieldDataElement,
   &VWAP_VOLUME_Element,
   "VWAP_VOLUME",
   0);

assert(VWAP_VOLUME_Element);
blpapi_Element_getValueAsFloat64(VWAP_VOLUME_Element,
   &vwap_volume,
   0);

printf("*security      =\%s\n", security);
printf("*sequenceNumber=\%d\n", sequenceNumber);
printf("*px_last       =\%f\n", px_last);
printf("*ds002         =\%s\n", ds002);
printf("*vwap_volume   =\%f\n", vwap_volume);
printf("\n");
}
}
blpapi_MessageIterator_destroy(iter);
}

static void handleOtherEvent(const blpapi_Event_t *event)
{
    blpapi_MessageIterator_t *iter    = 0;
    blpapi_Message_t         *message = 0;

    assert(event);

    printf("EventType=\%d\n", blpapi_Event_eventType(event));

    iter = blpapi_MessageIterator_create(event);
    assert(iter);

    while (0 == blpapi_MessageIterator_next(iter, &message)) {
        blpapi_CorrelationId_t  correlationId;
        blpapi_Element_t       *messageElements = 0;

        assert(message);
        correlationId = blpapi_Message_correlationId(message, 0);
        printf("correlationId=\%d \%d \%lld\n",
               correlationId.valueType,
               correlationId.classId,
               correlationId.value.intValue);

        printf("messageType=\%s\n", blpapi_Message_typeString(message));

        messageElements = blpapi_Message_elements(message);
        assert(messageElements);
        blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
if (BLPAPI_EVENTTYPE_SESSION_STATUS ==
  blpapi_Event_eventType(event)
  && 0 == strcmp("SessionTerminated",
        blpapi_Message_typeString(message))){
    fprintf(stdout,
        "Terminating: %s\n",
        blpapi_Message_typeString(message));
    exit(1);
}

blpapi_MessageIterator_destroy(iter);

int main()
{
  blpapi_SessionOptions_t *sessionOptions = 0;
  blpapi_Session_t *session = 0;
  blpapi_CorrelationId_t requestId;
  blpapi_Service_t *refDataSvc = 0;
  blpapi_Request_t *request = 0;
  blpapi_Element_t *elements = 0;
  blpapi_Element_t *securitiesElements = 0;
  blpapi_Element_t *fieldsElements = 0;
  blpapi_CorrelationId_t correlationId;
  int continueToLoop = 1;

  sessionOptions = blpapi_SessionOptions_create();
  assert(sessionOptions);

  blpapi_SessionOptions_setServerHost(sessionOptions, "localhost");
  blpapi_SessionOptions_setServerPort(sessionOptions, "8194");

  session = blpapi_Session_create(sessionOptions, 0, 0, 0);
  assert(session);

  blpapi_SessionOptions_destroy(sessionOptions);

  if (0 != blpapi_Session_start(session)) {
    fprintf(stderr, "Failed to start session.\n");
    blpapi_Session_destroy(session);
    return 1;
  }

  if (0 != blpapi_Session_openService(session,"//blp/refdata")){
    fprintf(stderr, "Failed to open service //blp/refdata.\n");
    blpapi_Session_destroy(session);
    return 1;
  }
}
memset(&requestId, '\0', sizeof(requestId));
requestId.size = sizeof(requestId);
requestId.valueType = BLPAPI_CORRELATION_TYPE_INT;
requestId.value.intValue = (blpapi_UInt64_t)1;

blpapi_Session_getService(session, &refDataSvc, "//blp/refdata");

blpapi_Service_createRequest(refDataSvc,
    &request,
    "ReferenceDataRequest");
assert(request);

elements = blpapi_Request_elements(request);
assert(elements);

blpapi_Element_getElement(elements,
    &securitiesElements,
    "securities",
    0);
assert(securitiesElements);

blpapi_Element_setValueString(securitiesElements,
    "AAPL US Equity",
    BLPAPI_ELEMENT_INDEX_END);
blpapi_Element_setValueString(securitiesElements,
    "IBM US Equity",
    BLPAPI_ELEMENT_INDEX_END);
blpapi_Element_setValueString(securitiesElements,
    "BLAHBLAHBLAH US Equity",
    BLPAPI_ELEMENT_INDEX_END);

blpapi_Element_getElement(elements, &fieldsElements, "fields", 0);
blpapi_Element_setValueString(fieldsElements,
    "PX_LAST",
    BLPAPI_ELEMENT_INDEX_END);
blpapi_Element_setValueString(fieldsElements,
    "DS002",
    BLPAPI_ELEMENT_INDEX_END);
blpapi_Element_setValueString(fieldsElements,
    "VWAP_VOLUME",
    BLPAPI_ELEMENT_INDEX_END);

memset(&correlationId, '\0', sizeof(correlationId));
correlationId.size = sizeof(correlationId);
correlationId.valueType = BLPAPI_CORRELATION_TYPE_INT;
correlationId.value.intValue = (blpapi_UInt64_t)1;

blpapi_Session_sendRequest(session, request, &correlationId, 0, 0, 0, 0);
while (continueToLoop) {
    blpapi_Event_t *event = 0;

    blpapi_Session_nextEvent(session, &event, 0);
    assert(event);
    switch (blpapi_Event_eventType(event)) {
        case BLPAPI_EVENTTYPE_RESPONSE: /* final event */
            continueToLoop = 0;            /* fall through */
        case BLPAPI_EVENTTYPE_PARTIAL_RESPONSE:
            handleResponseEvent(event);
            break;
        default:
            handleOtherEvent(event);
            break;
    }
    blpapi_Event_release(event);
}

blpapi_Session_stop(session);

blpapi_Request_destroy(request);
blpapi_Session_destroy(session);

return 0;
Request Response Multiple Output

```
EventType=2
correlationId=0 0 0
messageType=SessionStarted
SessionStarted = {
}

EventType=9
correlationId=0 0 0
messageType=ServiceOpened
ServiceOpened = {
    *security      = AAPL US Equity
    *sequenceNumber=0
    *px_last       = 90.910000
    *ds002         = APPLE INC
    *vwap_volume   = 7603357.000000

    *security      = IBM US Equity
    *sequenceNumber=1
    *px_last       = 91.180000
    *ds002         = INTL BUSINESS MACHINES CORP
    *vwap_volume   = 3272079.000000

    *security      = BLAHBLAHBLAH US Equity
    securityError = {
        source = 161::bbdbs2
        code = 15
        category = BAD_SEC
        message = Unknown/Invalid security [nid:161]
        subcategory = INVALID_SECURITY
    }
}
```
F.5  Subscription Multiple

```c
/* SubscriptionMultiple.c */

#include <blpapi_correlationid.h>
#include <blpapi_element.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_request.h>
#include <blpapi_session.h>
#include <blpapi_subscriptionlist.h>

#include <assert.h>
#include <stdio.h>
#include <string.h>  /* for memset(3C) */
#include <unistd.h>  /* for pause(2) */

static int streamWriter(const char* data, int length, void *stream)
{
    assert(data);
    assert(stream);
    return fwrite(data, length, 1, (FILE *)stream);
}

typedef struct UserData {
    const char *d_label;
    FILE       *d_stream;
} UserData_t;

static void dumpEvent(const blpapi_Event_t *event,
                       const UserData_t     *userData)
{
    blpapi_MessageIterator_t *iter    = 0;
    blpapi_Message_t         *message = 0;

    assert(event);
    assert(userData);
    assert(userData->d_label);
    assert(userData->d_stream);

    fprintf(userData->d_stream, "handler label=%s\n", userData->d_label);
    fprintf(userData->d_stream, "eventType=%d\n",
            blpapi_Event_eventType(event));

    iter = blpapi_MessageIterator_create(event);
    assert(iter);

    for (message = blpapi_MessageIterator_begin(iter); message;
         message = blpapi_MessageIterator_next(message))
    {
        printf("eventType=%s\n", blpapi_Message_eventType(message));
        printf("eventTime=%s\n", blpapi_Message_eventTime(message));
        printf("symbol=%s\n", blpapi_Message_symbol(message));
        printf("symbolType=%d\n", blpapi_Message_symbolType(message));
        printf("exchange=%s\n", blpapi_Message_exchange(message));
        printf("exchangeType=%d\n", blpapi_Message_exchangeType(message));
        printf("description=%s\n", blpapi_Message_description(message));
        printf("productType=%s\n", blpapi_Message_productType(message));
    }
    blpapi_MessageIterator_destroy(iter);
    blpapi_MessageIterator_destroy(message);
}
```
while (0 == blpapi_MessageIterator_next(iter, &message)) {
    blpapi_CorrelationId_t correlationId;
    blpapi_Element_t *messageElements = 0;

    assert(message);
    printf("messageType=%s\n", blpapi_Message_typeString(message));
    messageElements=blpapi_Message_elements(message);
    correlationId = blpapi_Message_correlationId(message, 0);
    printf("correlationId=%d %d %lld\n",
            correlationId.valueType,
            correlationId.classId,
            correlationId.value.intValue);

    blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
}

static void handleDataEvent(const blpapi_Event_t *event,
                            const blpapi_Session_t *session,
                            const UserData_t *userData) {
    assert(event);
    assert(userData);
    fprintf(userData->d_stream, "handleDataEventHandler: enter\n");
    dumpEvent(event, userData);
    fprintf(userData->d_stream, "handleDataEventHandler: leave\n");
}

static void handleStatusEvent(const blpapi_Event_t *event,
                               const blpapi_Session_t *session,
                               const UserData_t *userData) {
    assert(event);
    assert(session);
    assert(userData); /* this application expects userData */
    fprintf(userData->d_stream, "handleStatusEventHandler: enter\n");
    dumpEvent(event, userData);
    fprintf(userData->d_stream, "handleStatusEventHandler: leave\n");
}

static void handleOtherEvent(const blpapi_Event_t *event,
                              const blpapi_Session_t *session,
                              const UserData_t *userData) {
    assert(event);
    assert(userData);
    assert(userData->d_stream);
fprintf(userData->d_stream, "handleOtherEventHandler: enter\n");
dumpEvent(event, userData);
fprintf(userData->d_stream, "handleOtherEventHandler: leave\n");
}

#ifdef __cplusplus
extern "C"
#endif

static void processEvent(blpapi_Event_t *event,
                         blpapi_Session_t *session,
                         void              *buffer)
{
    UserData_t *userData = (UserData_t *)buffer;

    assert(event);
    assert(session);
    assert(buffer);

    switch (blpapi_Event_eventType(event)) {
    case BLPAPI_EVENTTYPE_SUBSCRIPTION_DATA:
        handleDataEvent(event, session, userData);
        break;
    case BLPAPI_EVENTTYPE_SESSION_STATUS:
    case BLPAPI_EVENTTYPE_SERVICE_STATUS:
    case BLPAPI_EVENTTYPE_SUBSCRIPTION_STATUS:
        handleStatusEvent(event, session, userData);
        break;
    default:
        handleOtherEvent(event, session, userData);
        break;
    }
}

int main()
{
    blpapi_SessionOptions_t *sessionOptions = 0;
    blpapi_Session_t        *session        = 0;

    UserData_t               userData       = { "myLabel", stdout };
    /* IBM */
    const char *topic IBM  = "IBM US Equity";
    const char *fields IBM[] = { "LAST_TRADE" };
    const char **options IBM = 0;
    int numFields IBM = sizeof(fields IBM)/sizeof(*fields IBM);
    int numOptions IBM = 0;

    /* GOOG */
    const char *topic GOOG = "/ticket/GOOG US Equity";
    const char *fields GOOG[] = { "BID", "ASK", "LAST_TRADE" };
    const char **options GOOG = 0;
    int numFields GOOG = sizeof(fields GOOG)/sizeof(*fields GOOG);
    int numOptions GOOG = 0;
/* MSFT */
const char  *topic_MSFT = "MSFT US Equity"; /* Note: Typo! */
const char  *fields_MSFT[] = { "LAST_PRICE" };
const char  *options_MSFT[] = { "interval=.5" };
int         numFields_MSFT = sizeof(fields_MSFT)/
sizeof(*fields_MSFT);
int        numOptions_MSFT = sizeof(options_MSFT)/
sizeof(*options_MSFT);

/* CUSIP 097023105 */
const char  *topic_097023105 =
"/cusip/097023105?fields=LAST_PRICE&interval=5.0";
const char  **fields_097023105 = 0;
const char  **options_097023105 = 0;
int         numFields_097023105 = 0;
int        numOptions_097023105 = 0;

setbuf(stdout, 0); /* DO NOT SHOW */

blpapi_CorrelationId_t subscriptionId_IBM;
blpapi_CorrelationId_t subscriptionId_GOOG;
blpapi_CorrelationId_t subscriptionId_MSFT;
blpapi_CorrelationId_t subscriptionId_097023105;

memset(&subscriptionId_IBM, '\0', sizeof(subscriptionId_IBM));
subscriptionId_IBM.size = sizeof(subscriptionId_IBM);
subscriptionId_IBM.valueType = BLPAPI_CORRELATION_TYPE_INT;
subscriptionId_IBM.value.intValue = (blpapi_UInt64_t)10;

memset(&subscriptionId_GOOG, '\0', sizeof(subscriptionId_GOOG));
subscriptionId_GOOG.size = sizeof(subscriptionId_GOOG);
subscriptionId_GOOG.valueType = BLPAPI_CORRELATION_TYPE_INT;
subscriptionId_GOOG.value.intValue = (blpapi_UInt64_t)20;

memset(&subscriptionId_MSFT, '\0', sizeof(subscriptionId_MSFT));
subscriptionId_MSFT.size = sizeof(subscriptionId_MSFT);
subscriptionId_MSFT.valueType = BLPAPI_CORRELATION_TYPE_INT;
subscriptionId_MSFT.value.intValue = (blpapi_UInt64_t)30;

memset(&subscriptionId_097023105, '\0',
sizeof(subscriptionId_097023105));
subscriptionId_097023105.size =
sizeof(subscriptionId_097023105);
subscriptionId_097023105.valueType = BLPAPI_CORRELATION_TYPE_INT;
subscriptionId_097023105.value.intValue = (blpapi_UInt64_t)40;

sessionOptions = blpapi_SessionOptions_create();
assert(sessionOptions);

blpapi_SessionOptions_setServerHost(sessionOptions, "localhost");
blpapi_SessionOptions_setServerPort(sessionOptions, "8194");
session = blpapi_Session_create(sessionOptions,
    &processEvent,
    0,
    &userData);

assert(session);

blpapi_SessionOptions_destroy(sessionOptions);

if (0 != blpapi_Session_start(session)) {
    fprintf(stderr, "Failed to start session.\n");
    blpapi_Session_destroy(session);
    return 1;
}

if (0 != blpapi_Session_openService(session,"//blp/mktdata")){
    fprintf(stderr, "Failed to open service //blp/mktdata.\n");
    blpapi_Session_destroy(session);
    return 1;
}

blpapi_SubscriptionList_t *subscriptions =
    blpapi_SubscriptionList_create();

blpapi_SubscriptionList_add(subscriptions,
    topic_IBM,
    &subscriptionId_IBM,
    fields_IBM,
    options_IBM,
    numFields_IBM,
    numOptions_IBM);

blpapi_SubscriptionList_add(subscriptions,
    topic_GOOG,
    &subscriptionId_GOOG,
    fields_GOOG,
    options_GOOG,
    numFields_GOOG,
    numOptions_GOOG);

blpapi_SubscriptionList_add(subscriptions,
    topic_MSFT,
    &subscriptionId_MSFT,
    fields_MSFT,
    options_MSFT,
    numFields_MSFT,
    numOptions_MSFT);

blpapi_SubscriptionList_add(subscriptions,
    topic_097023105,
    &subscriptionId_097023105,
    fields_097023105,
    options_097023105,
    numFields_097023105,
    numOptions_097023105);
blpapi_Session_subscribe(session, subscriptions, 0, 0, 0);

pause();

blpapi_SubscriptionList_destroy(subscriptions);
blpapi_Session_destroy(session);

return 0;
}
handleStatusEventHandler: enter
director label=myLabel
eventType=2
messageType=SessionStarted
correlationId=0 0 0
SessionStarted = {
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
director label=myLabel
eventType=9
messageType=ServiceOpened
correlationId=0 0 0
ServiceOpened = {
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
director label=myLabel
eventType=3
messageType=SubscriptionFailure
correlationId=1 0 30
SubscriptionFailure = {
  reason = {
    errorCode = 2
    description = Invalid security
category = BAD_SEC
    source = BBDB@n151
  }
}
handleStatusEventHandler: leave
handleStatusEventHandler: enter
director label=myLabel
eventType=3
messageType=SubscriptionStarted
correlationId=1 0 40
SubscriptionStarted = {
  exceptions[] =
}
messageType=SubscriptionStarted
correlationId=1 0 10
SubscriptionStarted = {
  exceptions[] =
}
messageType=SubscriptionStarted
correlationId=1 0 20
SubscriptionStarted = {
  exceptions[] =
}
handleStatusEventHandler: leave
handleDataEventHandler: enter
handler label=myLabel
eventType=8
messageType=MarketDataEvents
correlationId=1 0 10
MarketDataEvents = {
  LAST_PRICE = 92.410000
  BID = 92.360000
  ASK = 92.390000
  VOLUME = 11337256
  HIGH = 93.200000
  LOW = 91.220000
  BEST_BID = 92.360000
  BEST_ASK = 92.390000
  LAST_TRADE = 92.410000
  OPEN = 92.130000
  IMBALANCE_BID = 92.390000
  IMBALANCE_ASK =
  ORDER_IMB_BUY_VOLUME = 44300.000000
  ORDER_IMB_SELL_VOLUME =
  VWAP = 92.213100
  THEO_PRICE = 0.000000
  LAST_ALL_SESSIONS = 92.410000
  IMBALANCE_INDIC_RT = BUY
  BID_ALL_SESSION = 92.030000
  ASK_ALL_SESSION = 92.370000
  TRADING_DT_REALTIME = 2009-02-05
  EQY_TURNOVER_REALTIME = 1042895294.262009
  NYSE_LRP_HIGH_PRICE_RT = 93.360000
  NYSE_LRP_LOW_PRICE_RT = 91.360000
  TOT_CALL_VOLUME_CUR_DAY_RT = 5625
  TOT_PUT_VOLUME_CUR_DAY_RT = 2314
  TOT_OPT_VOLUME_CUR_DAY_RT = 7939
  PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
  IN_AUCTION_RT = false
  RT_API_MACHINE = p142
  ALL_PRICE_SIZE = 1200
  ALL_PRICE = 92.379200
  ALL_PRICE_COND_CODE =
  BID_COND_CODE =
  ASK_COND_CODE =
  VOLUME_THEO = 0
  LAST_AT_TRADE_TDY = 0.000000
  SIZE_LAST_AT_TRADE_TDY = 0
  OPEN_YLD_TDY = 0.000000
  HIGH_YLD_TDY = 0.000000
  LOW_YLD_TDY = 0.000000
  LAST_YLD_TDY = 0.000000
  MID_TDY = 0.000000
  SIZE_LAST_TRADE_TDY = 579500
  IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 92.130000
ASK_SIZE_TDY = 79
BID_SIZE_TDY = 5
VOLUME_TDY = 11337256
LAST_PRICE_TDY = 92.410000
BID_TDY = 92.360000
ASK_TDY = 92.390000
HIGH_TDY = 93.200000
LOW_TDY = 91.220000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 92.410000
LAST_DIR = 1
LAST2_DIR = 1
BID_DIR = 1
ASK_DIR = 1
BID2 = 92.360000
ASK2 = 92.390000
SIZE_LAST_TRADE = 579500
ASK_SIZE = 79
BID_SIZE = 5
API_MACHINE = p142
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 579500
IS_DELAYED_STREAM = false
EID = 14003
PREV_SES_LAST_PRICE = 92.780000
RT_PX_CHG_NET_1D = -0.369900
RT_PX_CHG_PCT_1D = -0.398684
TIME = 21:00:27.000+00:00
LAST_UPDATE_BID_RT = 21:00:22.000+00:00
LAST_UPDATE_ASK_RT = 21:00:22.000+00:00
NYSE_LRP_SEND_TIME_RT = 20:59:57.000+00:00
BID_ASK_TIME = 21:00:22.000+00:00
SES_START = 14:30:00.000+00:00
SES_END = 21:30:00.000+00:00
}
handleDataEventHandler: leave
handleDataEventHandler: enter
handler label=myLabel
eventType=8
messageType=MarketDataEvents
correlationId=1 0 10
MarketDataEvents = {
    LAST_PRICE = 92.410000
    BID = 92.360000
    ASK = 92.390000
    VOLUME = 11337256
    BEST_BID = 92.360000
    BEST_ASK = 92.390000
}
LAST_TRADE = 92.410000
IMBALANCE_BID = 92.390000
IMBALANCE_ASK =
ORDER_IMB_BUY_VOLUME = 44300.000000
ORDER_IMB_SELL_VOLUME =
VWAP = 92.251200
THEO_PRICE = 92.390000
LAST_ALL_SESSIONS = 92.410000
IMBALANCE_INDIC_RT = BUY
BID_ALL_SESSION = 92.030000
ASK_ALL_SESSION = 92.370000
EQY_TURNOVER_REALTIME = 1042895294.262009
NYSE_LRP_HIGH_PRICE_RT = 93.360000
NYSE_LRP_LOW_PRICE_RT = 91.360000
TOT_CALL_VOLUME_CUR_DAY_RT = 5625
TOT_PUT_VOLUME_CUR_DAY_RT = 2314
TOT_OPT_VOLUME_CUR_DAY_RT = 7939
PUT_CALL_VOLUME_RATIO_CUR_DAY_RT = 0
IN_AUCTION_RT = false
ALL_PRICE_SIZE = 1200
ALL_PRICE = 92.379200
ALL_PRICE_COND_CODE =
VOLUME_THEO = 545600
LAST_AT_TRADE_TDY = 0.000000
SIZE_LAST_AT_TRADE_TDY = 0
OPEN_YLD_TDY = 0.000000
HIGH_YLD_TDY = 0.000000
LOW_YLD_TDY = 0.000000
LAST_YLD_TDY = 0.000000
MID_TDY = 0.000000
SIZE_LAST_TRADE_TDY = 579500
IND_BID_FLAG = false
IND_ASK_FLAG = false
OPEN_TDY = 92.130000
ASK_SIZE_TDY = 79
BID_SIZE_TDY = 5
VOLUME_TDY = 11337256
LAST_PRICE_TDY = 92.410000
BID_TDY = 92.360000
ASK_TDY = 92.390000
HIGH_TDY = 93.200000
LOW_TDY = 91.220000
BID_YLD_TDY = 0.000000
ASK_YLD_TDY = 0.000000
LAST2_PRICE = 92.410000
LAST_DIR = 1
LAST2_DIR = 1
BID_DIR = 1
ASK_DIR = 1
BID2 = 92.360000
ASK2 = 92.390000
SIZE_LAST_TRADE = 579500
ASK_SIZE = 79
BID_SIZE = 5
EXCH_CODE_LAST =
EXCH_CODE_BID =
EXCH_CODE_ASK =
TRADE_SIZE_ALL_SESSIONS_RT = 579500
IS_DELAYED_STREAM = false
EID = 14003
RT_PX_CHG_NET_1D = -0.369900
RT_PX_CHG_PCT_1D = -0.398684
TIME = 21:00:27.000+00:00
LAST_UPDATE_BID_RT = 21:00:22.000+00:00
LAST_UPDATE_ASK_RT = 21:00:22.000+00:00
NYSE_LRP_SEND_TIME_RT = 20:59:57.000+00:00
BID_ASK_TIME = 21:00:22.000+00:00