Bloomberg

Bloomberg API

Version 3.x

Developer's Guide

March 21, 2013
Version 2.47

Bloomberg

Table of Contents

Bloomberg

Bloomberg

Bloomberg

Bloomberg

Bloomberg

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

Version	Date	Description of Changes
2.0	11/05/09	This is the first release of the Bloomberg API Developer's Guide.
2.34	01/27/12	Updated license notice on front page.
2.35	02/21/12	Added "Case Sensitivity" on page 77.
2.38	31/12/12	Updated "IntradayTickRequest: Sequence" on page 168, "Intra-dayTickResponse: Choice" on page 170, and "Reference Data Service Response Values" on page 180.
2.39	5/17/12	Added "Managed B-Pipe" on page 115.
2.40	7/19/12	Added "Overview" on page 115 and "Source Reference Service" on page 151.
2.41	10/03/12	Corrected items in Table 9-4, "Chain Subservice Examples," on page 139 and Table 9-4, "Chain Subservice Examples," on page 139.
2.42	11/14/12	Updated "IntradayTickResponse: Choice" on page 170.
2.43	12/21/12	Updated "IntradayBarRequest: Sequence" on page 172.
2.44	01/04/13	Added footnote to Table 9-14, "Enumeration Values," on page 153 and updated Table 9-4, "Chain Subservice Examples," on page 139.
2.45	01/14/13	Updated "Managed B-Pipe" on page 115.
2.46	01/29/13	Added "Instruments Service" on page 99. Updated MD_BOOK_TYPE table on page 122.
2.47	03/21/13	Updated MD_BOOK_TYPE table on page 122 and Notes on page 130.

Bloomberg

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 Managed B-PIPE or 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.

Bloomberg

Managed 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.

Bloomberg

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.

Bloomberg

1.1.1 Features

Feature	Details
Four Languages, One Interface	API 3.0 provides all new programming interfaces in: ● Java ● C ● C++ ● .Net 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.
Lightweight Interfaces	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. It is now possible to get the maximum performance out of the Java, .Net, C, and C++ versions of the interface.
Extensible Service-Oriented Data Model	The new API generically understands the notions of subscription and request-response services. 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. 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.
Field Level Subscriptions	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. 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. For example, if quotes are of no interest to an application, processing and bandwidth consumption can be cut by as much as 90%.

Bloomberg

Feature	Details
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.

Feature	Details
Thread-Safe	All language bindings for the new API are now fully thread-safe. Applications can safely process responses and make requests simultaneously from multiple threads of execution.
32- and 64-bit Programming Support	The Java and .Net API work on both 32- and 64-bit platforms. The C and C++ API libraries come in a 32-bit version with a 64- bit version coming in the future.
Pure Java Implementation	The Java API is implemented entirely in Java. Bloomberg did not use JNI to wrap either our existing C library or the new C++ library.
Fully Introspective data model	An application can discover a service and its attributes at runtime.
Simplified Permissioning Model	Release 3.0 of the Server API provides a simplified permissioning model that allows you to simply provide a user's UUID and IP address. The API returns the permissions to you.

The Bloomberg API is the interface to the following Bloomberg products:

- The Bloomberg Platform
- Managed 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 Managed B-PIPE

Managed B-PIPE leverages the Bloomberg distribution platform and managed entitlements system. Managed B-PIPE allows clients to connect applications providing solutions that work with client proprietary and 3rd party applications. Managed 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).

Figure 1-2: Managed B-PIPE

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).

Figure 1-3: The Desktop API

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).

Figure 1-4: The Server API: User Mode

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.

Bloomberg

Figure 1-5: The Server API: Server Mode

1.2 The Programming Examples

The Bloomberg API is provided as Java, .Net, C++, and C libraries. The libraries share the same object model, class and method names, and programming paradigm to make it easy for developers to switch languages. In this document, Java is used for the sample code and for the programming interface specification.

Complete, contiguous listings of the Java code examples are provided in "Java Examples" on page 216 and the programming interface specification is found in "Schemas" on page 116.

For the sample programs in the other supported languages see:

- ".Net Examples" on page 248
- "C++ Examples" on page 275
- "C Examples" on page 301

1.3 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).

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.

1.4 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 21.
- 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 30, "Requests and Responses" on page 38, and "Subscriptions" on page 46.

Bloomberg

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.

```
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 76 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 `Event`s 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 `Event`s 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:

```
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);
    }

.........
```

Bloomberg

```
… …
        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
            case Event.EventType.Constants.PARTIAL_RESPONSE:
                handleResponseEvent(event);
                break;
            default:
                 handleOtherEvent(event);
                break;
            }
        }
    }
```

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:

    ```
        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.

```
    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 `Message`s 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:

```
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 `Event`s of type neither `PARTIAL_RESPONSE` nor `RESPONSE`.

```
EventType=SESSION_STATUS
correlationID=null
messageType=SessionStarted
SessionStarted = {
}
EventType=SERVICE_STATUS
correlationID=Internal: 1
messageType=ServiceOpened
ServiceOpened = {
}
```

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.

```
    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.

Bloomberg

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_1D = -4.29
    RT_PX_CHG_PCT_1D = -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.

```
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.

Bloomberg

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:

```
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.

```
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.

```
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.

```
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:

```
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.

```
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 {
            Handle error.
        }
    }
    break;
}
```

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.

```
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`.

```
case Event.EventType.Constants.PARTIAL_RESPONSE: {
    dumpEvent(event); // Handle Partial Response
    break;
}

case Event.EventType.Constants.RESPONSE: {
    dumpEvent(event); // Handle final response

    // Example complete; shut-down.
    try {
        session.stop(Session.StopOption.ASYNC);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("terminate process from handler");
    System.exit(0);
    break;
}
```

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 228.

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

- `RequestResponseMultiple.cs` ("Request Response Multiple" on page 263)

- `RequestResponseMultiple.cpp` ("Request Response Multiple" on page 288)

- `RequestResponseMultiple.c` ("Request Response Multiple" on page 321)

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:

Bloomberg

- 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:

```
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`".

```
………
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.

```
………
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 `Element`s occurs so frequently that the `Request` class provides convenience methods that are more concise (but also obscure the `Element` sub-structure):

```
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

Bloomberg

schema, but is also conveniently viewed, as we have done earlier, by printing the response in the response event handler code.

```
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.

```
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`.

```
  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")) {
        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.getElementAsString("security");
        int     sequenceNumber =
                          securityData.getElementAsInt32("sequenceNumber");
        if (securityData.hasElement("securityError")) {
          Element securityError =
                             securityData.getElement("securityError");
          handle error
          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.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("");
        }
      }
    }
  }
```

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.

```
* 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.

Bloomberg

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 The Programming Example

The example explored in this chapter is `SubscriptionMultiple.java`. A complete listing of this example and its output can be found in .

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

- `SubscriptionMultiple.cs` ()
- `SubscriptionMultiple.cpp` ()
- `SubscriptionMultiple.c` ()

5.2 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:

```
using SubscriptionList =
        System.Collections.Generic.List<Bloomberglp.Blpapi.Subscription>;
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:

```
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:

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

Bloomberg

5.3 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.

See for more details on the "`//blp/mktdata`" service.

```
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_TURNOVER_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
}
```

Bloomberg

5.4 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 created earlier with the correlation ID of `subscriptionID_IBM` we can use the following code fragment:

```
………
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.

5.5 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.

```
………
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`.

Bloomberg

5.6 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.

5.7 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.

5.8 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 Managed B-Pipe requires explicit permission for access.

5.9 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.

Bloomberg

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.

Bloomberg

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 Managed 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 Managed 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 Managed B-PIPE, revolves around the use of a class called the `Identity`. These objects represent a user (or an application in Managed 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 Managed 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.

Bloomberg

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 automaticaly 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.

Bloomberg

Figure 6-1: Server API: User Mode: Authorization by IP Address

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.

Figure 6-2: Server API: Server Mode: Authorization by IP Address

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.

Bloomberg

```java
<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.
```

```
  for (boolean continueToLoop = true; continueToLoop; )
  {
   Event event = session.nextEvent();
   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;
   }
  }
  .........
```

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;
 }
```

Bloomberg

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 Managed B-PIPE Authorization

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

Managed 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 Managed 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

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

Figure 6-3: Obtaining a User's Identity in Managed B-PIPE

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<GO>** 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:

```
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 <GO>**.

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.

```
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.

Bloomberg

```cpp
<C++>

// 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
```

Bloomberg

```
for (MessageIterator messageIterator(tokenEvent);
        messageIterator.next(); )
{
Message message = messageIterator.message();
if (TOKEN_FAILURE == message.messageType())
{
    std::cerr << "Failed to obtain token" << std::endl;
    return 1;
}
assert(TOKEN_SUCCESS == message.messageType());
token.assign(message.getElementAsString("token"));
break;
}

………authorization stage………
```

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

Managed 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 Managed B-PIPE.

6.5 Authorization

For Managed 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 Managed B-PIPE.

```cpp
<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………
```

Bloomberg

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 1:

EID	Description	Source	Examples
14005	NASDAQ Level 1	NASDAQ	MSFT UQ Equity,
INTC UQ Equity[a]			
[b]	BGN	Bloomberg Generic	CT2@BGN Govt
23599	U.S. Treasures	Merrill Lynch	CT2@ML Govt
14014, 14076[c]	London Stock Exchange Level 1 & 2	LSE	VOD LN Equity

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.

Bloomberg

First, the data request must be modified to request that entitlement identifiers be included with the returned data. For example:

```Java
<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);
………
```

Bloomberg

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

```Java
<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.getElementAsString("security");
        int sequenceNumber =
        securityData.getElementAsInt32("sequenceNumber");
        if (securityData.hasElement("securityError"))
        {
        handle error
        }
        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
                }
            }
        }
    }
```

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 allow`MultipleCorrelationsPerMsg` 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.

Bloomberg

For example,

```Java
<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
<Java>

………Extract 'securityData' from response message………

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.

Bloomberg

6.7 Specific Application Types (Managed B-PIPE only)

Managed B-PIPE introduced the concepts of Named Applications. These are setup on **EMRS <GO>** 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.

Bloomberg

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 Managed 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_blbg_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 Managed 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.

7 Core Services

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 Managed B-Pipe-only services, please see "Managed B-Pipe Services" on page 115

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.

Bloomberg

Syntax

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.

Govt	Corp	Mtge
M-Mkt	Muni	Pfd
Equity	Comdy	Index
Curncy	Client	

Case Sensitivity

- The API will adjust the yellow key (Equity, Cmdty, 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 lN" 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.

Bloomberg

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 `Curncy` Yellow Key use a space instead of `@` to separate the security from the pricing source, for example, "`GBPUSD BAAM Curncy`".

In Managed B-Pipe, if you request a security with a specific pricing source (e.g., CT10@BGN Govt), and if there is not pricing available on the specified source, you will receive a 'Pricing Not Available' error. Reference data requests that are not pricing specific will return data. In the event that you are not entitled for the requested source, you will receive a 'Not Entitled' error and no data will be returned.

In Desktop API and Server API, if you request a security with a specific pricing source and if there is not pricing available on the specified source, Bloomberg systems will hunt through a list of preferred sources to find pricing and return that pricing to you.

Note that the logic described above only applies to 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.

Bloomberg

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).

Bloomberg

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`:

```
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`:.

```
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.getElement("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.

```
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");
```

Bloomberg

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

```
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), interval and date/time start and end-points in UTC must be specified.

This example shows how to construct an `IntradayTickRequest`:

```
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

is influenced by the date and time range of your request and the level of market activity during that period.

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

  for (int i = 0; i < numItems; ++i) {
    Element  item  = data.getValueAsElement(i);
    Datetime time  = item.getElementAsDate("time");
    String   type  = item.getElementAsString("type");
    double   value = item.getElementAsFloat64("value");
    int      size  = item.getElementAsInt32("size");
    String   cc;
    if (item.hasElement("conditionCodes")) {
      cc = item.getElementAsString("conditionCodes");
    }
    Process values
  }
}
```

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<GO>** 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`.

```
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));
```

Response Overview

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`.

```
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 **PRTU<GO>** 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.

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".

7.3 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.

```
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.susbcribe (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.susbcribe(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 provides streaming (real time and delayed) intraday bars. This service provides the functionality to obtain intraday bars for trade volume, number of ticks, open, close, high, low and time of last trade. 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.

For example: `"//blp/mktbar/ticker/VOD LN Equity"`
`"//blp/mktbar/isin/GB00B16GWD56 LN"`

The only field that can be submitted for this service is `LAST_PRICE`. The following code snippet shows a subscription to market bars: .

```
Assume that the blp/mktbar service has already been opened successfully.
SubscriptionList  d_subscriptions = new SubscriptionList();
d_subscriptions.add("//blp/mktbar/ticker/VOD LN Equity","LAST_PRICE",
                    "interval=5",CorrelationId(1));
d_session.subscribe(d_subscriptions);
```

Response Behavior

There are three types of messages that can occur in a `SUBSCRIPTION_DATA` event. The first event received is `MarketBarStart`. This occurs at every new bar; therefore the frequency of this will depend upon the interval setting. A `MarketBarStart` will return all fields ("A.4 Market Bar Subscription" on page 195). Subsequently, on every last price update a MarketBarUpdate will be sent. This will only include fields that have updated since the bar

start or last update. Fields that are always updated are `VOLUME, NUMBER_OF_TICKS,` `TIME` and `CLOSE. MarketBarEnd` only occurs when the last market bar has been received - i.e., the end_time has been reached. This message only contains `TIME`.

Please note there is no initial summary returned for streaming intraday bars, a reference data request or a subscription will be required to get an initial snapshot if required.

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`.

```
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");
```

Bloomberg

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`.

```
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.getElement("fieldInfo");
    System.out.println(
      fieldData.getElementAsString("id") + " " +
      fieldInfo.getElementAsString("mnemonic") + " (" +
      fieldInfo.getElementAsString("description") + ") " +
      fieldInfo.getElementAsString("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`.

```
Service fieldInfoService = session.getService("//blp/apiflds");
Request request         =
fieldInfoService.createRequest("FieldSearchRequest");
request.set("searchSpec", "last price");
Element exclude = request.getElement("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 <u>"Appendix A Schemas"</u>. This example shows how to construct a `CategorizedFieldSearchRequest`.

```
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`.

```
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.getElement("fieldInfo");
      System.out.println(
        fieldData.getElementAsString("id") + " " +
        fieldInfo.getElementAsString("mnemonic") + " (" +
        fieldInfo.getElementAsString("description") + ") " +
        fieldInfo.getElementAsString("datatype"));
      }
    }
  }
}
```

Bloomberg

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:

String	Rows Specified
"1"	The first row on the page
"1,2,3"	Rows 1,2 and 3 on the page
"1,6-10,15,16"	Row 1, rows 6 to 10 and rows 15 and 16

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.

```
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.

```
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.

Bloomberg

```
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

Bloomberg

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 7-1: Data Type Description Table

	Description
Historical End of Day	End-of-day data for a specified period of time in increments of days, weeks, months, quarters, or years.
Intraday	Intraday data for a specified period of time in increments of minutes. Based on Bid, Ask, or Trade events, data such as open, high, low, close, and volume can be retrieved for the interval of time specified.
Real-time	Real-time data and events.

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.

```
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("dataRange").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").
                      GetElement("dmiStudyAttributes");
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.

```
    private void processResponseEvent(Message msg)
    {
            Element security = msg.GetElement(SECURITY_NAME);
            string ticker = security.GetValueAsString();
            System.Console.WriteLine("\nTicker: " + ticker);
            if (security.HasElement("securityError"))
            {
                printErrorInfo("\tSECURITY FAILED: ",
                    security.GetElement(SECURITY_ERROR));
                continue;
            }
            Element fields = msg.GetElement(STUDY_DATA);
            if (fields.NumValues > 0)
            {
                int numValues = fields.NumValues;
                for (int j = 0; j < numValues; ++j)
                {
                    Element field = fields.GetValueAsElement(j);
                    for (int k = 0; k < field.NumElements; k++)
                    {
                        Element element = field.GetElement(k);
                      System.Console.WriteLine("\t" + element.Name + " = " +
                            element.GetValueAsString());
                    }
                    System.Console.WriteLine("");
                }
            }
    }
```

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.

Bloomberg

```
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");
```

Response Behaviour

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

```
private void processResponseEvent(Message msg)
{
        Element security = msg.GetElement(SECURITY_NAME);
        string ticker = security.GetValueAsString();
        System.Console.WriteLine("\nTicker: " + ticker);
        if (security.HasElement("securityError"))
        {
            printErrorInfo("\tSECURITY FAILED: ",
                security.GetElement(SECURITY_ERROR));
            continue;
        }
        Element fields = msg.GetElement(STUDY_DATA);
        if (fields.NumValues > 0)
        {
            int numValues = fields.NumValues;
            for (int j = 0; j < numValues; ++j)
            {
                Element field = fields.GetValueAsElement(j);
                for (int k = 0; k < field.NumElements; k++)
                {
                    Element element = field.GetElement(k);
                  System.Console.WriteLine("\t" + element.Name + " = " +
                        element.GetValueAsString());
                }
            }
        }
}
```

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 hasbeen successfully opened.

```
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.susbcribe (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 53.

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.

Request	Operation
Security Lookup Request	InstrumentListRequest Operation
Curve Lookup Request	CurveListRequest Operation
Government Lookup Request	GovtListRequest Operation

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.

```
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.

```
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.

```
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:

```
  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:" << 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("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

```
   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

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.

Bloomberg

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.`

```
// BroadcastOneTopic.cpp
… …
int main()
{
    SessionOptions sessionOptions;
    sessionOptions.setServerHost("platform");
    sessionOptions.setServerPort(8195);

sessionOptions.setAuthenticationOptions("AuthenticationType=OS_LOGON");
    MyEventHandler myEventHandler;

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

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 details.

Bloomberg

```cpp
        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);
```

Bloomberg

```
   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");
        }
        }
        }
    }
  }
```

8.3.3 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`".

.

```
     … …
    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));
        }
    }

  … …
```

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.

```
     … …
     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;
}
```

Note: The standard C library 'sleep' function is used above. The argument specifies the number of seconds to sleep.

8.4 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

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.

```cpp
// 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.

```
    … …  create 'activePublication' collection, the managing mutex,
          and the event handler … …
    … …  create 'session' and obtain 'Identity'… …

    const char *myService = "//blp/mktdata8";
    if (!session.registerService(myService, providerIdentity)) {
        std::cerr <<"Failed to register " << myService << std::endl;
        return -1;
    }
 … …
  }
```

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.

Bloomberg

```cpp
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);

                    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.

```
    Service service = session.getService(myService);

    Name messageType("MyMessageType");
    Name fieldName("MyFieldName");
    for (int value = 1; true; ++ value, sleep(10)) {
        pthread_mutex_lock(&activePublicationsMutex);

        if (0 == activePublications.size()) {
            continue;
        }

        Event event = service.createPublishEvent();
        EventFormatter eventFormatter(event);
        for (Publications::iterator iter = activePublications.begin();
                                    iter != activePublications.end();
                                    ++iter) {
            const std::string& cusip = iter->second;
            eventFormatter.appendMessage(messageType, iter->first);
            eventFormatter.setElement(fieldName, myValueFor(cusip,
value));
        }
        pthread_mutex_unlock(&activePublicationsMutex);

        session.publish(event);
    }

    session.stop();

    return 0;
}
```

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.

Bloomberg

9 Managed 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 Managed 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 Managed B-Pipe users, please see .

Important Notice

Field filtering is available as a configuration option, which means that Managed 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 Managed 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.

Bloomberg

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.

Mnemonic	Description
BEST_BID1 thru BEST_BID10	First thru tenth best bid price in ten levels of market depth
BEST_BID1_SZ thru BEST_BID10_SZ	Size of first thru tenth best bid in ten levels of market depth
BEST_ASK1 thru BEST_ASK10	First thru tenth best ask price in ten levels of market depth
BEST_ASK1_SZ thru BEST_ASK10_SZ	Size of first thru tenth best ask in ten levels of market depth

For further information regarding making a subscription, please read the .

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 Managed 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.

Bloomberg

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 dectection 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.

Key Field	Format	Example
Bloomberg Symbol	//blp/mktdepth/bsym/source/symbol	//blp/mktdepth/bsym/LN/VOD?type=MBL //blp/mktdepth/bsym/US/AAPL?type=MBO
Ticker	//blp/mktdepth/ticker/symbol	//blp/mktdepth/ticker/ESM2 Index?type=MBL
BSID	//blp/mktdepth/bsid/bsid	//blp/mktdepth/bsid/399432473346?type=MBO
Bloomberg Unique ID	//blp/mktdepth/buid/source/identifier	//blp/mktdepth/buid/US/EQ0010080100001000?type=MBL
SEDOL	//blp/mktdepth/sedol/source/identifier	//blp/mktdepth/sedol/US/2005973?type=MBL
CUSIP	//blp/mktdepth/cusip/source/identifier	//blp/mktdepth/cusip/US/459200101?type=MBL
ISIN	//blp/mktdepth/isin/source/identifier	//blp/mktdepth/isin/US/US4592001014?type=MBL
Parsekeyable	//blp/mktdepth/bpkbl/identifier	//blp/mktdepth/bpkbl/QCZ1 Index?type=MBL
Bloomberg Global ID	//blp/mktdepth/bbgid /source/bbgid	//blp/mktdepth/bbgid/UP/BBG000BH2658?type=MBO

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.

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

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

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.

Bloomberg

The possible string values, which indicate what action should be taken in response to the market depth event, are listed in the table below.

Name	Value	Description
UNASSIGNED	0	The default constant 'UNASSIGNED' is used to initialize all enumeration type fields
ADD	1	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 in the event and market depth orders or levels inferior to event passed to one position inferior. For example, if a new order is added to position one of the market depth table, then the previous order at position one is shifted to position two. The order at position two is shifted to position three and so on until you get to the market depth window size. If the ADD results in Bid or ASK sides to have more levels than the value configured in MB[LO]_WINDOW_SIZE, the last level in the corresponding side should be dropped. It will be up to you to cache MB[LO]_WINDOW_SIZE from the Initial paint event to handle this scenario.
DEL	2	Delete this event from the market depth cache. The delete should occur at the position passed in the market depth event. When cached market event at the position passed in the delete is removed, all position inferior should have their positions shifted by one. For example, if position one is deleted from a market by order or market by price event, the position two becomes one, position three becomes two, etc.
DELALL	3	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.
DELBETTER	4	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 in that it deletes the current order, where the EXEC command modifies the current order.
DELSIDE	5	Delete all events on the corresponding side (bid/ask) of the order book.
EXEC	7	Trade Execution. Find the corresponding order in the cache, replace event details with this event and then delete any prior superior orders.
MOD	8	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 the cached event by the fields and data in the new event.
REPLACE	10	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 and size will be sent when there is no active price or order at this level.

Name	Value	Description
REPLACE_BY_BROKER	11	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.
CLEARALL	12	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.
REPLACE_CLEAR	13	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.

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.

Book Type (MD_BOOK_TYPE)	Initial Paint Table Command (MD_TABLE_CMD_RT)	Table Commands in Real-Time Messages (MD_TABLE_CMD_RT)
MBO[L]-AMD	ADD	CLEARALL, ADD, MOD, DELETE, DELSIDE, DELBETTER, EXEC
MBO[L]-RBL	REPLACE	CLEARALL, REPLACE, REPLACE_CLEAR
MBO-RBB	REPLACE_BY_BROKER	CLEARALL, REPLACE_BY_BROKER, REPLACE_CLEAR

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 "MarketDepthSubscriptionSnaphotExample" 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.

Message::Fragment Type Enumerators	
FRAGMENT_NONE	Message is not fragmented
FRAGMENT_START	The first fragmented message
FRAGMENT_INTERMEDIATE	Intermediate fragmented messages
FRAGMENT_END	The last fragmented message

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.

```
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);
        d_orderBooks[Side::BIDSIDE].book_type = d_orderBooks[Side::ASKSIDE].book_type;
        // clear cache
        d_orderBooks[Side::ASKSIDE].doClearAll();
        d_orderBooks[Side::BIDSIDE].doClearAll();
    }
}
```

Figure 9-3: Checking for the Fragment Type (C++)

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_BID_POSITION_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 = 2012-05-25T19:53:55.000+00:00
        MBL_TABLE_ASK[] = {
        }
        MBL_TABLE_BID[] = {
        }
        MBO_TABLE_ASK[] = {
        }
        MBO_TABLE_BID[] = {
        }
    }
```

Bloomberg

```
    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.

Data Response For Request-By-Broker (RBB) Order Books

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")

```
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[] = {
              }
    }
```

```
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[] = {
            }
    }
```

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

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, Managed 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 Managed 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, Managed B-Pipe will automatically initiate the recap without any action by the client application.

When Managed 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 Managed 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 9-1: Fields Affected by Recaps

Fields	Descriptions
MKTDEPTH_EVENT_SUBTYPE	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 "BID_RETRANS" or "ASK_RETRANS".
MBL_SEQNUM_RT and MBO_SEQNUM_RT	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.
MD_TABLE_CMD_RT	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 "DELSIDE" 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.

Table 9-1: Fields Affected by Recaps

Fields	Descriptions
MD_MULTI_TICK_UPD_RT	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.

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 POSITON=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.

Bloomberg

Table 9-2: Market List String Definitions

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

a. See examples in Table 9-4 on page 139.

Bloomberg

Code Examples

You will find two separate examples in the Managed 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

Managed 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 Managed 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 Managed 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 Managed 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 Managed 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 Managed B-Pipe. It is possible that an index or curve list member is not available on Managed 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 Managed 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 9-3: Default Security Class of List Members

Underlying Security Class	Default Chain Member Security Class
Currency	Option
Equity	Option
Fixed Income	N/A
Fund	Option
Future Root	Future
Future Contract	Option
Index	Members
Option	N/A
Warrant	N/A
Curve	Members

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.

Bloomberg

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

Type of Chain List	Example Subscription String	Topic Type	Topic Key[a]	Refreshes[b]
Option Chains	//blp/mktlist/chain/bsym/LN/BP/	/bsym	/<DX282>/<DY003>	No
	//blp/mktlist/chain/bsid/678605350316	/bsid	/<ID122>	No
	//blp/mktlist/chain/buid/LN/EQ0010160500001000	/buid	/<DX282>/<ID059>	No
	//blp/mktlist/chain/bbid/LN/EQ0010160500001000	/bbid	/<DX194>	No
	//blp/mktlist/chain/bpkbl/BP/LN Equity	/bpkbl	/<DX282>/<EX005>	No
	//blp/mktlist/chain/esym/LN/BP	/esym	/<DX282>/<ID032>	No
	//blp/mktlist/chain/cusip/UN/594918104	/cusip	/<DX282>/<ID032>	No
	//blp/mktlist/chain/isin/LN/GB00B16GWD56	/isin	/<DX282>/<ID005>	No
	//blp/mktlist/chain/sedol/UN/2588173	/sedol	/<DX282>/<ID002>	No
	//blp/mktlist/chain/bbgid/UP/BBG000BH2658	/bbgid	/<ID135>	No
	//blp/mktlist/chain/ticker/VOD LN Equity	/ticker	/<DX194>/<DS587>	No
Index List	//blp/mktlist/chain/bsym/FTUK/UKX Index;secclass=Option	/bsym	/<DX282>/<DY003>	Daily
Yield Curve	//blp/mktlist/chain/bpkbl/YCMM0010 Index	/bpkbl	/<identifier>	Daily
GDCO	//blp/mktlist/chain/gdco/broker/id	/gdco	/<broker_id>/<mon_id>	N/A
EID List	//blp/mktlist/chain/eid/14014	/eid	/<source>	No
Source List	//blp/mktlist/chain/source/UN;secclass=Equity	/source	/<source>	No

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:

FLDS <GO> Identifier	Mnemonic	FLDS <GO> Identifier	Mnemonic
DX194	PARSEKYABLE_DES_SOURCE	ID005	ID_ISIN
DX282	FEED_SOURCE	ID032	ID_CUSIP
DY003	ID_BB_SEC_NUM_DES	ID059	ID_BB_UNIQUE
EX005	ID_EXCH_SYMBOL	ID122	ID_BB_SEC_NUM_SRC
ID002	ID_SEDOL1	ID035	ID_BB_GLOBAL

Table 9-5: Additional "chain" Subscription Examples

Subscription String	Returns
//blp/mktlist/chain/bsym/FTUK/UKX Index;secclass=Option	Returns options on the UKX Index
//blp/mktlist/chain/bsym/FTUK/UKX Index;secclass=Option&source=LN	Returns options on the UKX Index traded on source LN
//blp/mktlist/chain/cusip/~/459200101	SubscriptionFailure: ErrorCode=2; Description=NOTUNIQUE; Category=BAD_SEC **Note:** NOTUNIQUE is returned because the security has more than one source and the request is ambiguous.
//blp/mktlist/chain/bsid/1086627109973	Options for IBM Equity
//blp/mktlist/chain/bsym/US/IBM;secclass=Future	Returns futures for Equity
//blp/mktlist/chain/bpkbl/YCMM0010 Index	GBP LIBOR Curve members (Yield Curve)
//blp/mktlist/chain/eid/38736	List of all currencies available on EID 38736
//blp/mktlist/chain/bsym/US/HP	Returns a chain of options for the composite equity HP
//blp/mktlist/chain/bsym/DJI/INDU Index	Returns a chain of the members of the index.
//blp/mktlist/chain/bsid/1086627109973	This resolves to currency (/IT/UBY) so will return an option chain.
//blp/mktlist/chain/isin/LN/GB00B16GWD56;secclass=Warrant	Returns a chain of warrants for the underlying instrument.
//blp/mktlist/chain/bsym/FTUK/UKX Index;secclass=Index	Returns a chain of members for the specified index identifier (equivalent to //blp/mktlist/chain/bsym/FTUK/UKX Index)
//blp/mktlist/chain/source/UN;secclass=Equity	Returns a list of Equities under source UN
//blp/mktlist/chain/bsym/BGN/YCCF0009 Index	Returns the list of members for the curve "YCCF0009 Index"
//blp/mktlist/chain/bsid/1086627109973	This resolves to currency (/IT/UBY) so will return an option chain.
//blp/mktlist/chain/bpkbl/IBM US Equity	Returns a chain of options (equivalent to //blp/mktlist/chain/bsid/399432473346; secclass=Option).
//blp/mktlist/chain/isin/LN/GB00B16GWD56;secclass=Warrant	Returns a chain of warrants for the underlying instrument.
//blp/mktlist/chain/bsym/eNYL/XG1;secclass=Future	Returns a chain of futures for the underlying instrument

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.

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

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

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":

Bloomberg

```
ListRecap = {
    LIST_ID = //blp/mktlist/chain/source/TQ
    EID = 35009
    LIST_LISTTYPE = Source List
    LIST_INSERT_ENTRIES[] =
        LIST_INSERT_ENTRIES = {
            ID_BB_SEC_NUM_SRC = 7992941317759
            FEED_SOURCE = TQ
            ID_BB_SEC_NUM_DES = RHI
            ID_BB_UNIQUE = EQ0000000006685436
            SECURITY_TYP2 = Equity
        }
        LIST_INSERT_ENTRIES = {
            ID_BB_SEC_NUM_SRC = 7992941317760
            FEED_SOURCE = TQ
            ID_BB_SEC_NUM_DES = GIL
            ID_BB_UNIQUE = EQ0000000006687052
            SECURITY_TYP2 = Equity
        }

...MORE...

        LIST_INSERT_ENTRIES = {
            ID_BB_SEC_NUM_SRC = 7992961685384
            FEED_SOURCE = TQ
            ID_BB_SEC_NUM_DES = ECONB
            ID_BB_UNIQUE = EQ0000000023559102
            SECURITY_TYP2 = Equity
        }
        LIST_INSERT_ENTRIES = {
            ID_BB_SEC_NUM_SRC = 7992961685385
            FEED_SOURCE = TQ
            ID_BB_SEC_NUM_DES = FIS1V
            ID_BB_UNIQUE = EQ0000000023561882
            SECURITY_TYP2 = Equity
        }
        LIST_INSERT_ENTRIES = {
            ID_BB_SEC_NUM_SRC = 7992961842174
            FEED_SOURCE = TQ
            ID_BB_SEC_NUM_DES = ENQ1
            ID_BB_UNIQUE = EQ0000000023716301
            SECURITY_TYP2 = Equity
        }
    LIST_ORDERED = NOTORDERED
    LIST_MUTABLE = MUTABLE
}
ListData = {
    LIST_ID = //blp/mktlist/chain/source/TQ
    EID = 35009
    LIST_ACTION = ADD_AND_END
    FEED_SOURCE = TQ
    ID_BB_SEC_NUM_DES = SNOP
}
```

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.

Table 9-7: Message::Fragment Type Enumerators

Enumerator	Description
FRAGMENT_NONE	Message is not fragmented
FRAGMENT_START	The first fragmented message
FRAGMENT_INTERMEDIATE	Intermediate fragmented messages
FRAGMENT_END	The last fragmented message

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 Managed B-Pipe mnemonic and FLDS <GO> field identifiers.

Table 9-8: Supported <Topic Type> Values

Topic Type	Topic Key	Managed B-Pipe Field	FLDS<GO> Field
/bpkbl	/<identifier>	PARSEKYABLE_DES_SOURCE	DX194 and DS587
/bsid	/<identifier>	ID_BB_SEC_NUM_SRC	ID122
/bsym	/<identifier>	ID_BB_SEC_NUM_DES	DY003
/buid	/<identifier>	ID_BB_UNIQUE	ID059
/cusip	/<identifier>	ID_CUSIP	ID032
/esym	/<identifier>	ID_EXCH_SYMBOL	EX005->EX011
/isin	/<identifier>	ID_ISIN	ID005
/sedol	/<identifier>	ID_SEDOL1	ID002
/bbgid	/<identifier>	ID_BB_GLOBAL	ID135
/ticker	/<identifier>	PARSEKYABLE_DES_SOURCE	DX194 and DS587

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

Key Field	Format	Result
Bloomberg Unique ID	//blp/mktlist/secids/buid/uniqueid	All instrument IDs for the given buid
	//blp/mktlist/secids/buid/EQ0010080100001000	
Bloomberg Symbol	//blp/mktlist/secids/bsym/symbol	All instrument IDs for the given bsym
	//blp/mktlist/secids/bsym/VOD	
SEDOL	//blp/mktlist/secids/sedol/sedol	All instrument IDs for the given SEDOL
	//blp/mktlist/secids/sedol/2005973	
CUSIP	//blp/mktlist/secids/cusip/cusip	All instrument IDs for the given CUSIP
	//blp/mktlist/secids/cusip/459200101	
ISIN	//blp/mktlist/secids/isin/isin	All instrument IDs for the given ISIN
	//blp/mktlist/secids/isin/US4592001014	
Parsekeyable	//blp/mktlist/secids/bpkbl/parsekeyable	All instrument IDs for the given parsekeyable
	//blp/mktlist/secids/bpkbl/UKX Index	

Bloomberg

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

Key Field	Format	Result
Message Scraping (MSG1)	//blp/mktlist/secids/bsym/MSGSCRP	The list of MSG1 instruments.
	//blp/mktlist/secids/bsym/MSGSCRP	
Bloomberg Global ID	//blp/mktlist/secids/bbgid/globalid	All instrument IDs for the given bbgid
	//blp/mktlist/secids/bbgid/BBG000BLNNH6	
Bloomberg Ticker	//blp/mktlist/secids/ticker/symbol	All instrument IDs for the given ticker
	//blp/mktlist/secids/ticker/IBM US Equity	

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

Subscription String	Returns
//blp/mktlist/secids/cusip/459200101	This example returns all IDs for the given CUSIP.
//blp/mktlist/secids/cusip/459200101;source=US	This example returns all IDs for the given CUSIP, but limited to source US.

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.

```
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);
```

Bloomberg

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 Managed 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;
               }
            }
         }
      }
   }
}
```

Bloomberg

```cpp
// return true if processing is completed, false otherwise
void processResponseEvent(Event event)
{
   MessageIterator msgIter(event);
   while (msgIter.next()) {
      Message msg = msgIter.message();
      Element responseCode;
      if ((msg.asElement().getElement(&responseCode, "responseCode") == 0) &&
         !responseCode.isNull())
      {
         int resultCode = responseCode.getElementAsInt32("resultCode");
         if (resultCode > 0)
         {
           std::string message = responseCode.getElementAsString("resultCode");
           std::string sourceId = responseCode.getElementAsString("sourceId");
           std::cout << "Request Failed: "<< message <<  std::endl;
           std::cout << "Source ID: " << sourceId << std::endl;
           std::cout << "Result Code: " << resultCode << std::endl;
           continue;
         }
      }

      Element snapshot = msg.getElement("snapshot");
      size_t numElements = snapshot.numElements();
      for (size_t i = 0; i < numElements; ++i)
      {
         const Element dataItem = snapshot.getElement(i);
         // Checking if the data item is Bulk data item
         if (dataItem.isArray()){
            processBulkData(dataItem);
         }else{
            std::cout << "\t" << dataItem.name() << " = " <<
               dataItem.getValueAsString() << std::endl;
         }
      }
   }
}
```

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.

```
□- Events
    ⊞- ListData
    ⊞- ListRecap
□- Operations
    □- SnapshotRequest
        □- Responses
            □- responseCode (SEQUENCE)
                ― resultCode (INT32)
                ― resultText (STRING)
                ― sourceId (STRING)
            □- snapshot (SEQUENCE)
                ― EID (INT32)
                ― LIST_DES (STRING)
                ― LIST_ID (STRING)
                ⊞- LIST_INSERT_ENTRIES (SEQUENCE)
                ― LIST_LISTTYPE (ENUMERATION)
                ― LIST_MUTABLE (ENUMERATION)
                ― LIST_ORDERED (ENUMERATION)
                ― LIST_UNDERLYING_FEED_SOURCE (STRING)
                ― LIST_UNDERLYING_ID_BSID (INT64)
                ― LIST_UNDERLYING_SECURITY_CLASS (STRING)
                ― LIST_UNDERLYING_SECURITY_TYP2 (STRING)
        □- SnapshotRequest (SEQUENCE)
            ― security (STRING)
```

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 resultCode, 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

<service owner>	For Managed B-Pipe is "blp"
<service name>	Source Reference and Tick Size subscription service name is "/srcref"
<subservice name>	/brokercodes, /conditioncodes, /tradingstatuses or /ticksizes (see Table 9-11.)
<topic type>	/eid
<topic key>	EID-Number (FEED_EID1 => FEED_EID4)

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

Table 9-12: Subservice Definitions

Subservice	Subscription String Format	Description
/brokercodes	//blp/srcref/brokercodes/eid/<eid>	List of all possible broker codes for a specified EID
/conditioncodes	//blp/srcref/conditioncodes/eid/<eid>	List of Market Depth, Quote, and Trade condition codes for a specified EID
/tradingstatuses	//blp/srcref/tradingstatuses/eid/<eid>	List of trading statuses and trading periods for a specified EID.
/ticksizes	//blp/srcref/ticksizes/eid/<eid>	List of Tick Sizes for a specified EID.

Filters can be used for /conditioncodes and /tradingstatuses subscription only. Here are the possible filters available for each:

Table 9-13: Filters for Events

Filter Name (type)	Subscription String Format
Subservice Name: /conditioncodes	
TRADE	//blp/srcref/conditioncodes/eid/<eid>?type=TRADE
QUOTE	//blp/srcref/conditioncodes/eid/<eid>?type=QUOTE
MKTDEPTH	//blp/srcref/conditioncodes/eid/<eid>?type= MKTDEPTH
TRADE,QUOTE	//blp/srcref/conditioncodes/eid/<eid>?type=TRADE,QUOTE
TRADE,MKTDEPTH	//blp/srcref/conditioncodes/eid/<eid>?type= TRADE,MKTDEPTH
QUOTE,MKTDEPTH	//blp/srcref/conditioncodes/eid/<eid>?type= QUOTE,MKTDEPTH
TRADE,QUOTE,MKTDEPTH	//blp/srcref/conditioncodes/eid/<eid>?type= TRADE,QUOTE,MKTDEPTH
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. Managed 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 Managed 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. Managed B-Pipe has introduced filters for some of its subservices to allow you to subscribe to the data you are most interested.

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

4. With Managed 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.

Bloomberg

Code Example

You will find a SourceRefSubscriptionExample in the Managed 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.

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

Name	Description	Values
SOURCE_REF_EVENT_TYPE_RT	This specifies the event type.	Possible enumeration values: DESCRIPTION BROKER_CODE TRADE_CONDITION_CODE QUOTE_CONDITION_CODE MKTDEPTH_CONDITION_CODE TRADING_PERIOD TRADING_STATUS TICK_SIZE_TABLE
SOURCE_REF_EVENT_SUBTYPE_RT	This specifies the event sub-type	Possible enumeration values: INITPAINT - Initial Paint REFRESH - Daily Refresh[a] UPDATE - Intraday Update

a. Refreshes are performed daily at approximately 6pm (Eastern Standard Time).

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

Bloomberg

(INITPAINT) and refresh sub-type messages. Table 9-15 will assist you in determining which SOURCE_REF_EVENT_TYPE_RT field types to expect based on the subservice in your subscription.

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 9-15: Event Types by Subservice

Subservice Name	Response Event Types
/brokercodes	DESCRIPTION + BROKER_CODE
/conditioncodes	DESCRIPTION + TRADE_COND_CODE + QUOTE_COND_CODE + MKTDEPTH_COND_CODE
/tradingstatuses	DESCRIPTION + TRADING_PERIOD + TRADING_STATUS
/ticksizes[a]	DESCRIPTION + TICK_SIZE_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 9-16: Event Type Fields

Field Name	Type	Contents
DESCRIPTIONS	SourceReferenceDescriptions	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.
BROKER_CODES	SourceReferenceBrokerCodes	Contains the feed EID and feed source, along with a list of BROKER_CODE entries containing each item's Bloomberg mnemonic and associated name.
TRADE_COND_CODES	SourceReferenceTradeConditionCodes	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.

Table 9-16: Event Type Fields

Field Name	Type	Contents
QUOTE_COND_CODES	SourceReferenceQuoteConditionCodes	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).
MKTDEPTH_COND_CODES	SourceReferenceMarketDepthConditionCodes	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).
TRADING_PERIODS	SourceReferenceTradingPeriods	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.
TRADING_STATUSES	SourceReferenceTradingStatuses	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.
TICK_SIZE_TABLES	TickSizeTable	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.

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

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

Message::Fragment Type Enumerators	
FRAGMENT_NONE	Message is not fragmented
FRAGMENT_START	The first fragmented message
FRAGMENT_INTERMEDIATE	Intermediate fragmented messages
FRAGMENT_END	The last fragmented message

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 = 0.050000
                            TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 0.000000
                            TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 10000000000.000000
                    }
        }
-- 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 --
}
```

Bloomberg

```
*****************************************************************
*   TICKSIZE INTRADAY UPDATE
*****************************************************************
SourceReferenceUpdates = {
    SOURCE_REF_EVENT_TYPE_RT = TICK_SIZE_TABLE
    SOURCE_REF_EVENT_SUBTYPE_RT = UPDATE
    EID = 35009
    TICK_SIZE_TABLES[] =
        TICK_SIZE_TABLES = {
            FEED_SOURCE = LN
            FEED_EID = 14014
            TICK_SIZE_TABLE_IDENTIFIER_RT = 5995
            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 = 0.300000
                    TICK_SIZE_TBL_BAND_LOWER_VAL_RT = 0.250000
                    TICK_SIZE_TBL_BAND_UPPER_VAL_RT = 100000000.000000
                }
        }
-- 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.

Bloomberg

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

A.2.1 Operations

Operation Name	Request Type	Response Type	Description
HistoricalDataRequest	HistoricalDataRequest	HistoricalDataResponse	Request Historical Data
IntraDayTickRequest	IntraDayTickRequest	IntraDayTickResponse	Request Intraday TIck Data
IntraDayBarRequest	IntraDayBarRequest	IntradayBarResponse	Request Intraday Bar Data
ReferenceDataRequest	ReferenceDataRequest	ReferenceDataResponse	Request Reference Data
PortfolioDataRequest	PortfolioDataRequest	PortfolioDataResponse	Request Portfolio Data
BeqsRequest	BeqsRequest	BeqsResponse	Request EQS Screen Data

A.2.2 ReferenceDataRequest: Sequence

Securities: A stock or bond.			
Element	**Element Value**	**Type**	**Description**
securities	string array	string	See "Security/Securities" on page 76 for additional details.
Example Syntax: `Element securities = request.GetElement("securities");` `securities.AppendValue("VOD LN Equity");`			

Bloomberg

Fields: the reference fields desired which correspond to data points. See **FLDS<GO>** for a list of more information.			
Element	**Element Value**	**Type**	**Description**
fields		string	See "Fields" on page 78 for additional details.

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

Overrides: Append overrides to modify the calculation			
Element	**Element Value**	**Type**	**Description**
fieldID		string	field mnemonic, PRICING_SOURCE, or field alpha-numeric, PR092. Review **FLDS<GO>** for list of possible overrides.
value		string	the desired override value

Example Syntax: `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.			
Element	**Element Value**	**Type**	**Description**
returnEids	TRUE or FALSE	Boolean	Setting this to true will populate fieldData with an extra element containing a name and value for the EID date.

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

Return Formatted Value: returns all data as a data type string			
Element	**Element Value**	**Type**	**Description**
returnFormattedValue	TRUE or FALSE	Boolean	Setting to true will force all data to be returned as a string.

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

Use UTC Time: return date and time values as Coordinated Universal Time (UTC) values			
Element	**Element Value**	**Type**	**Description**
useUTCTime	TRUE or FALSE	Boolean	Setting to true returns values in UTC. Setting this to false will default to the **TZDF<GO>** settings of the requestor.

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

Forced Delay: returns the latest reference data up to the delay period.			
Element	**Element Value**	**Type**	**Description**
forcedDelay	TRUE or FALSE	Boolean	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.

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

![Bloomberg]

A.2.3 ReferenceDataResponse: Choice

Figure A-1 provides the structure of a ReferenceDataResponse. See "Reference Data Service Response" on page 179 for more information.

Figure A-1: Reference Data Request Response

Bloomberg

A.2.4 HistoricalDataRequest: Sequence

Securities: A stock or bond.			
Element	**Element Value**	**Type**	**Description**
securities		string	See "Security/Securities" on page 76 for additional details.
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.			
Element	**Element Value**	**Type**	**Description**
fields		string array	See "Fields" on page 78 for additional details.
Example Syntax: `Element fields = request.GetElement("fields");` `fields.AppendValue("PX_LAST");`			

Start Date: the first date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
startDate	yyyymmdd	string	The start date in a year/month/day format.
Example Syntax: `request.Set("startDate", "20090601");`			

End Date: the end date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
endDate	yyyymmdd	string	The end date in a year/month/day format. This will default to the current day if not specified.
Example Syntax: `request.Set("endDate", "20100601");`			

Period Adjustment: Determine the frequency and calendar type of the output. To be used in conjunction with Period Selection.			
Element	**Element Value**	**Type**	**Description**
periodicityAdjustment	ACTUAL	string	These revert to the actual date from today (if the end date is left blank) or from the End Date
	CALENDAR	string	For pricing fields, these revert to the last business day of the specified calendar period. Calendar Quarterly (CQ), Calendar Semi-Annually (CS) or Calendar Yearly (CY).
	FISCAL	string	These periods revert to the fiscal period end for the company - Fiscal Quarterly (FQ), Fiscal Semi-Annually (FS) and Fiscal Yearly (FY) only
Example Syntax: `request.Set("periodicityAdjustment", "ACTUAL");`			

Bloomberg

Period Selection: Determine the frequency of the output. To be used in conjunction with Period Adjustment.			
Element	**Element Value**	**Type**	**Description**
periodicitySelection	DAILY	string	Returns one data point per day
	WEEKLY	string	Returns one data point per week
	MONTHLY	string	Returns one data point per month
	QUARTERLY	string	Returns one data point per quarter
	SEMI_ANNUALLY	string	Returns one data point per half year
	YEARLY	string	Returns one data point per year

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

Currency: Amends the value from local to desired currency			
Element	**Element Value**	**Type**	**Description**
currency	Currency of the ISO code, e.g., USD, GBP	string	The 3 letter ISO code. View **WCV<GO>** on the BLOOMBERG PROFESSIONAL service for a list of currencies.

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

Override Options: Indicates whether to use the average or the closing price in quote calculation.			
Element	**Element Value**	**Type**	**Description**
overrideOption	OVERRIDE_OPTION_CLOSE	string	Use the closing price in quote calculation
	OVERRIDE_OPTION_GPA	string	Use the average price in quote calculation

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).			
Element	**Element Value**	**Type**	**Description**
pricingOption	PRICING_OPTION_PRICE	string	Set quote to price
	PRICING_OPTION_YIELD	string	Set quote to yield

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.			
Element	**Element Value**	**Type**	**Description**
nonTradingDayFillOption	NON_TRADING_WEEKDAYS	string	Include all weekdays (Monday to Friday) in the data set
	ALL_CALENDAR_DAYS	string	Include all days of the calendar in the data set returned
	ACTIVE_DAYS_ONLY	string	Include only active days (days where the instrument and field pair updated) in the data set returned

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.			
Element	**Element Value**	**Type**	**Description**
nonTradingDayFillMethod	PREVIOUS_VALUE	string	Search back and retrieve the previous value available for this security field pair. The search back period is up to one month.
	NIL_VALUE	string	Returns blank for the "value" value within the data element for this field.

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

Max Data Points: the maximum number of data points to return.			
Element	**Element Value**	**Type**	**Description**
maxDataPoints		integer	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.

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

Return Entitlements: returns the entitlement identifiers associated with security.			
Element	**Element Value**	**Type**	**Description**
returnEids	TRUE or FALSE	Boolean	Setting this to TRUE will populate fieldData with an extra element containing a name and value for the EID date.

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

Return Relative Date: returns data with a relative date.			
Element	**Element Value**	**Type**	**Description**
returnRelativeDate	TRUE or FALSE	Boolean	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

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

Adjustment Normal: Adjust for "change on day"			
Element	**Element Value**	**Type**	**Description**
adjustmentNormal	TRUE or FALSE	Boolean	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.

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

Bloomberg

Adjustment Abnormal: Adjusts for Anormal Cash Dividends			
Element	**Element Value**	**Type**	**Description**
adjustmentAbnormal	TRUE or FALSE	Boolean	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.
Example Syntax: `request.Set("adjustmentAbnormal", true);`			

Adjustment Split: Capital Changes Defaults			
Element	**Element Value**	**Type**	**Description**
adjustmentSplit	TRUE or FALSE	Boolean	Adjust historical pricing and/or volume to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, Rights Offerings/Entitlement.
Example Syntax: `request.Set("adjustmentSplit", true);`			

Adjustment Follow DPDF: Follow the BLOOMBERG PROFESSIONAL service function **DPDF<GO>**			
Element	**Element Value**	**Type**	**Description**
adjustmentFollowDPDF	TRUE or FALSE	Boolean	Setting to true will follow the **DPDF<GO>** BLOOMBERG PROFESSIONAL service function. True is the default setting for this option.
Example Syntax: `request.Set("adjustmentFollowDPDF", true);`			

CalendarCodeOverride: Returns the data based on the calendar of the specified country, exchange, or religion.			
Element	**Element Value**	**Type**	**Description**
calendarCodeOverride	CDR <GO> calendar type	String	Returns the data based on the calendar of the specified country, exchange, or religion from **CDR<GO>**. 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.
Example Syntax: `request.Set("calendarCodeOverride", "US");`			

CalendarOverridesInfo: Returns data based on the calendar code of multiple countries, exchanges, or religious calendars from CDR<GO>.			
Element	**Element Value**	**Type**	**Description**
calendarOverrides	CDR <GO> calendar type	String array	Accepts a two-character calendar code null-terminated string of multiple country, exchange, or religious calendars from **CDR<GO>**. This will cause the data to be aligned according to the set calendar(s) including their calendar holidays. Only applies to DAILY requests.
calendareOverrides Operation	CDR_AND	String	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.
	CDR_OR	String	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.
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.			
Element	**Element Value**	**Type**	**Description**
fieldID		string	Specify a field mnemonic or alpha-numeric, such as PR092 or PRICING_SOURCE. Review **FLDS<GO>** for list of possible overrides.
value		string	The desired override value
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 179 for more information.
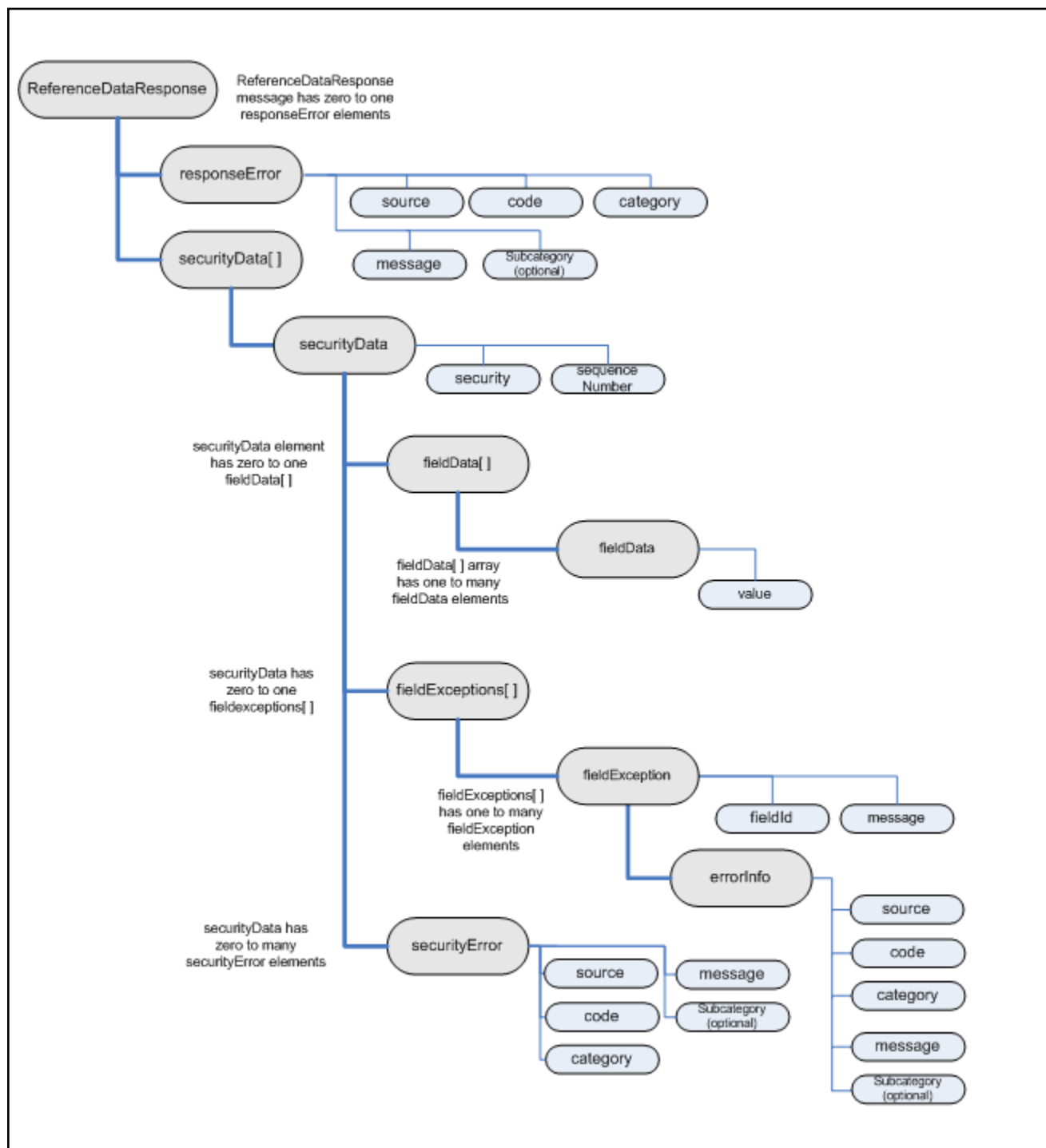

Figure A-2: Historical Data Response

A.2.6 IntradayTickRequest: Sequence

Securities: A stock or bond.			
Element	**Element Value**	**Type**	**Description**
securities		string	See "Security/Securities" on page 76 for additional details.
Example Syntax: `Element securities = request.GetElement("securities"); request.Set("security", "VOD LN Equity");`			

Start Date: the first date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
startDateTime	yyyy-mm-dd Thh:mm:ss	string	The start date and time.
Example Syntax: `request.Set("startDateTime", "2010-04-27T15:55:00");`			

End Date: the end date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
endDateTime	yyyy-mm-dd Thh:mm:ss	string	The end date and time.
Example Syntax: `request.Set("endDateTime", "2010-04-27T16:00:00");`			

Event Type: The requested data event type			
Element	**Element Value**	**Type**	**Description**
eventType	TRADE	string	Corresponds to LAST_PRICE
	BID	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	ASK	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
	BID_BEST	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	ASK_BEST	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
	MID_PRICE	string	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.
	AT_TRADE	string	Automatic trade for London Sets stocks.
	BEST_BID	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	BEST_ASK	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
Example Syntax: `request.Set("eventType", "TRADE");`			

Bloomberg

Include Condition Codes: return any condition codes that may be associated to a tick, which identifies extraordinary trading and quoting circumstances.			
Element	**Element Value**	**Type**	**Description**
includeConditionCodes	TRUE or FALSE	Boolean	A comma delimited list of exchange condition codes associated with the event. Review **QR<GO>** for more information on each code returned.
Example Syntax: `request.Set("includeConditionCodes", true);`			

Include Non Plottable Events: return ticks in the response that have condition codes			
Element	**Element Value**	**Type**	**Description**
includeNonPlottable Events	TRUE or FALSE	Boolean	Returns all ticks, including those with condition codes.
Example Syntax: `request.Set("includeNonPlottableEvents", true);`			

Include Exchange Codes: return the exchange code of the trade			
Element	**Element Value**	**Type**	**Description**
includeExchangeCodes	TRUE or FALSE	Boolean	The exchange code where this tick originated. Review **QR<GO>** for more information.
Example Syntax: `request.Set("includeExchangeCodes", true);`			

Return Entitlements: returns the entitlement identifiers associated with security.			
Element	**Element Value**	**Type**	**Description**
returnEids	TRUE or FALSE	Boolean	Option on whether to return EIDs for the security.
Example Syntax: `request.Set("returnEids", true);`			

Include Broker Codes: return the broker code of the trade			
Element	**Element Value**	**Type**	**Description**
includeBrokerCodes	TRUE or FALSE	Boolean	The broker code for Canadian, Finnish, Mexican, Philippine, and Swedish equities only. The Market Maker Lookup screen, **MMTK<GO>**, displays further information on market makers and their corresponding codes.
Example Syntax: `request.Set("includeBrokerCodes", true);`			

Include Reporting Party Side Codes: return transaction codes			
Element	**Element Value**	**Type**	**Description**
includeRpsCodes	TRUE or FALSE	Boolean	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).
Example Syntax: `request.Set("includeRpsCodes", true);`			

Bloomberg

Include Bank/Market Identifier Codes: return bank or market identifier code			
Element	**Element Value**	**Type**	**Description**
includeBicMicCodes	TRUE or FALSE	Boolean	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.
Example Syntax: `request.Set("includeBicMicCodes", true);`			

A.2.7 IntradayTickResponse: Choice

Figure A-3 provides the structure of an Intraday Tick Response. See for more information.

![Bloomberg]

Figure A-3: IntradayTickResponse

Bloomberg

A.2.8 IntradayBarRequest: Sequence

Securities: A stock or bond.			
Element	**Element Value**	**Type**	**Description**
securities		string	See "Security/Securities" on page 76 for additional details.
Example Syntax: `Element securities = request.GetElement("securities");` `request.Set("security", "VOD LN Equity");`			

Start Date: the first date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
startDateTime	yyyy-mm-dd Thh:mm:ss	string	The start date and time.
Example Syntax: `request.Set("startDateTime", "2010-04-27T15:55:00");`			

End Date: the end date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
endDateTime	yyyy-mm-dd Thh:mm:ss	string	The end date and time.
Example Syntax: `request.Set("endDateTime", "2010-04-27T16:00:00");`			

Event Type: The requested data event type			
Element	**Element Value**	**Type**	**Description**
eventType	TRADE	string	Corresponds to LAST_PRICE
	BID	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	ASK	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
	BID_BEST	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	ASK_BEST	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
	BEST_BID	string	Depending on the exchange bid ticks will be returned as BID, BID_BEST or BEST_BID.
	BEST_ASK	string	Depending on the exchange ask ticks will be returned as ASK, ASK_BEST or BEST_ASK.
Example Syntax: `request.Set("eventType", "TRADE");`			

Interval: the length of each bar returned			
Element	**Element Value**	**Type**	**Description**
interval	1...1440	integer	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.
Example Syntax: `request.Set("interval", 60);`			

Bloomberg

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.			
Element	**Element Value**	**Type**	**Description**
returnEids	TRUE or FALSE	Boolean	Option on whether to return EIDs for the security.

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

Adjustment Normal: Adjust "change on day"			
Element	**Element Value**	**Type**	**Description**
adjustmentNormal	TRUE or FALSE	Boolean	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.

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

Adjustment Abnormal: Adjust for Abnormal Cash Dividends			
Element	**Element Value**	**Type**	**Description**
adjustmentAbnormal	TRUE or FALSE	Boolean	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.

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

Adjustment Split: Capital Changes Defaults			
Element	**Element Value**	**Type**	**Description**
adjustmentSplit	TRUE or FALSE	Boolean	Adjust historical pricing and/or volume to reflect: Spin-Offs, Stock Splits/Consolidations, Stock Dividend/Bonus, Rights Offerings/ Entitlement.

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

Adjustment Follow DPDF: Follow the BLOOMBERG PROFESSIONAL service function **DPDF<GO>**			
Element	**Element Value**	**Type**	**Description**
adjustmentFollowDPDF	TRUE or FALSE	Boolean	Setting to true will follow the **DPDF<GO>** BLOOMBERG PROFESSIONAL service function. True is the default setting for this option..

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 179 for more information.
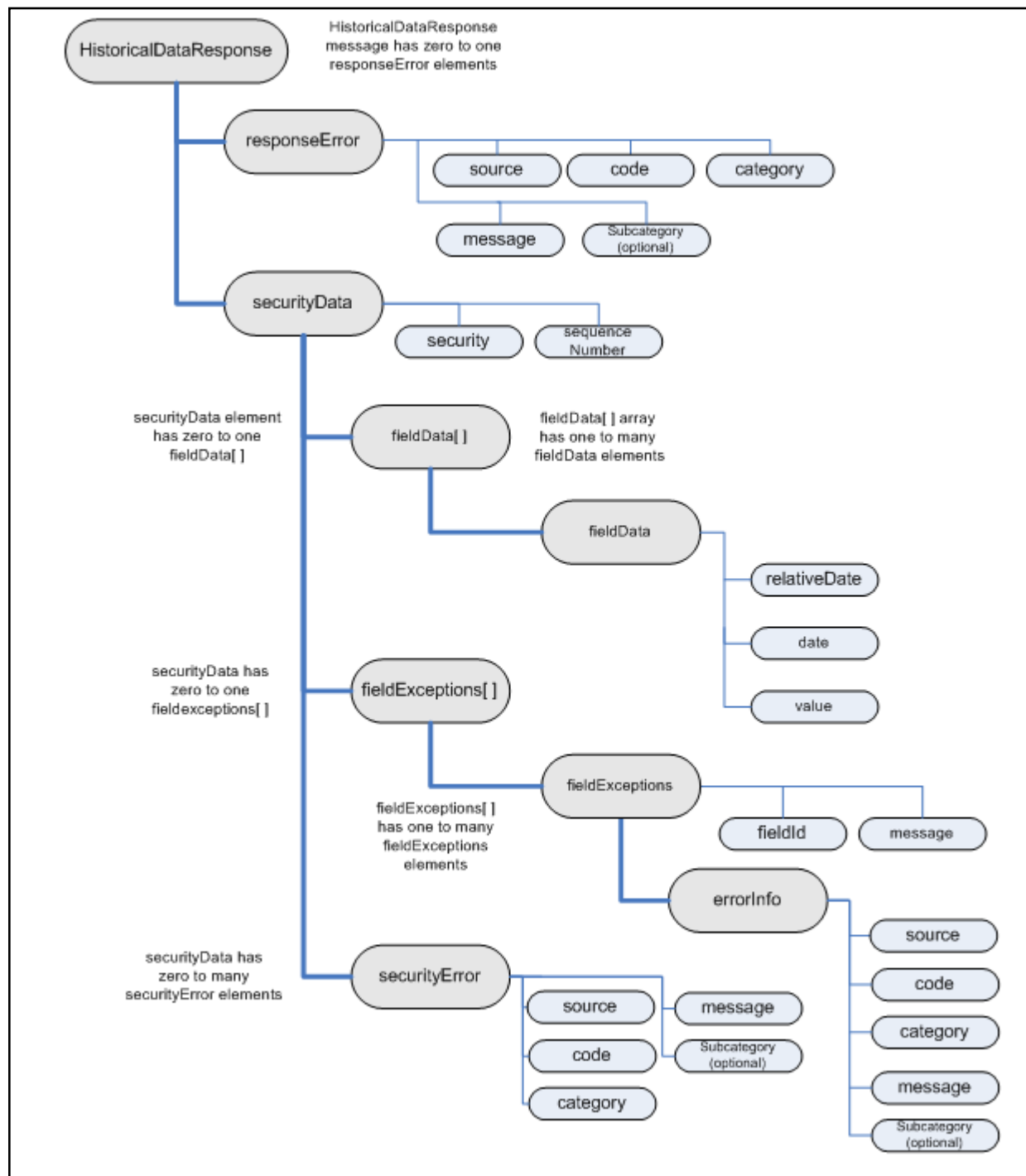

Figure A-4: IntradayBarResponse

A.2.10 PortfolioDataRequest: Sequence

Securities: A Portfolio ID			
Element	Element Value	Type	Description

Bloomberg

securities	string array	string	The users portfolio is identified by it's Portfolio ID, which can be found on the upper right hand corner of the settings tab on the portfolio's **PRTU<GO>** page on the BLOOMBERG PROFESSIONAL service.

Example Syntax: `Element securities = request.GetElement("securities");`
`securities.AppendValue("UXXXXXXX-X Client");`

Fields: The desired reference fields.

Element	Element Value	Type	Description
fields		string	The fields that can be used are PORTFOLIO_MEMBER PORTFOLIO_MPOSITION, PORTFOLIO_MWEIGHT & PORTFOLIO_DATA.

Example Syntax: `Element fields = request.GetElement("fields");`
`fields.AppendValue("PORTFOLIO_MEMBER ");`

Overrides: The Portfolio information can also be accessed historically by using the REFERENCE_DATE override field by supplying the date in 'yyyymmdd' format.

Element	Element Value	Type	Description
fieldId		string	Field mnemonic "REFERENCE_DATE"
value		string	The date in 'yyyymmdd' format.

Example Syntax: `Element overrides = request["overrides"];`
`Element override1 = overrides.AppendElement();`
`override1.SetElement("fieldId", "REFERENCE_DATE");`
`override1.SetElement("value", "20100111");`

Bloomberg

A.2.11 PortfolioDataResponse: Choice

Figure A-5 provides the structure of a PortfolioDataResponse. See "Reference Data Service Response" on page 179 for more information.

Figure A-5: Portfolio Data Request Response

Bloomberg

A.2.12 BEQSRequest: Sequence

screenName: An EQS screen name			
Element	**Element Value**	**Type**	**Description**
screenName	string	string	(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 <GO>** on the BLOOMBERG PROFESSIONAL service.
Example Syntax: `request.Set("screenName", "Global Volume Surges");`			
screenType: Screen Type.			
Element	**Element Value**	**Type**	**Description**
screenType	PRIVATE or GLOBAL	string	Use PRIVATE for user-defined EQS screen. Use GLOBAL for Bloomberg EQS screen.
Example Syntax: `request.Set("screenType", "GLOBAL");`			
languageId: Specify the language for field names to be returned for screen data			
Element	**Element Value**	**Type**	**Description**
languageId (optional)		string	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
Example Syntax: `request.Set("languageId", "FRENCH");`			
Group: Specify group name.			
Element	**Element Value**	**Type**	**Description**
Group (optional)		string	Screen folder name here as defined in **EQS<GO>**.
Example Syntax: `request.Set("Group", "Global Emerging Markets");`			

Bloomberg

A.2.13 BEQSResponse: Choice

Figure A-1 provides the structure of a BEQSResponse. See "Reference Data Service Response" on page 179 for more information.
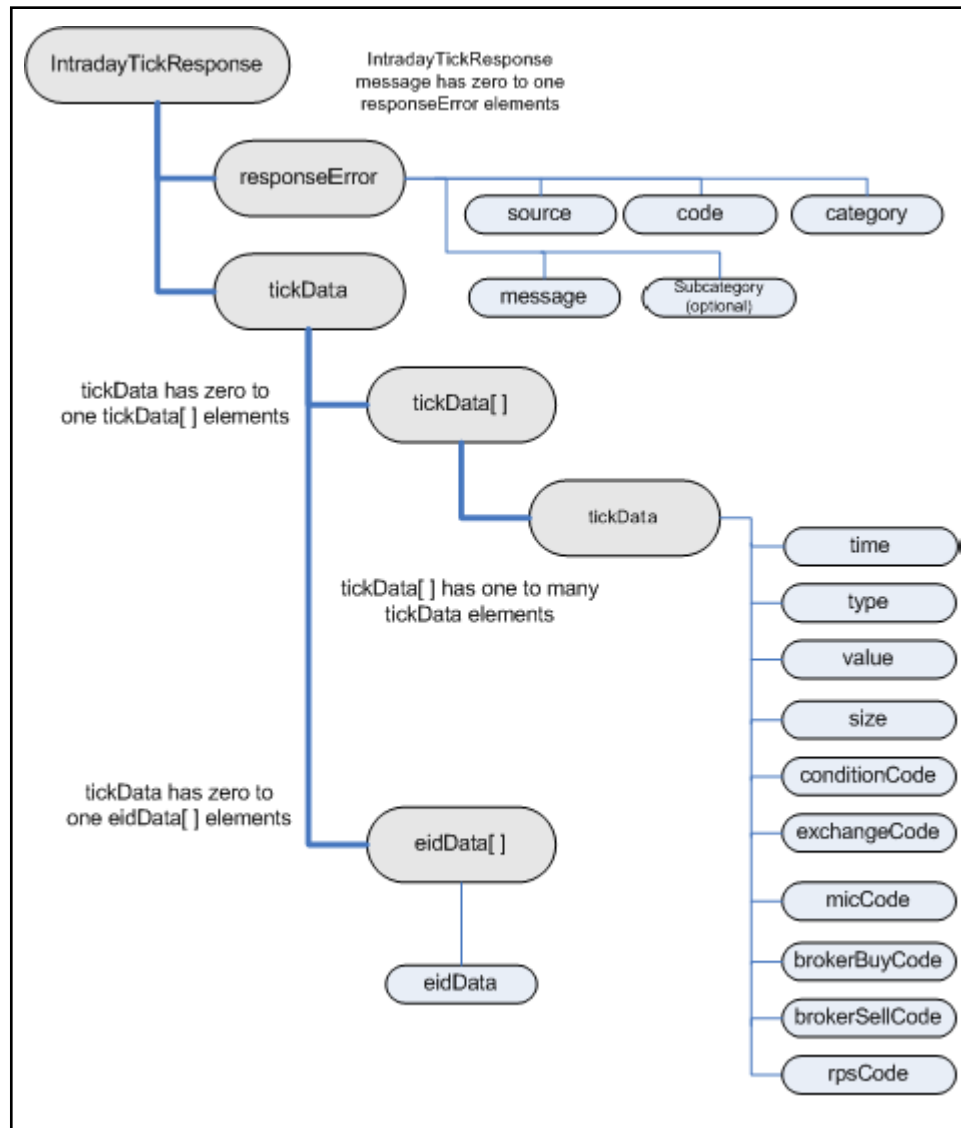

Figure A-6: BEQS Response

Bloomberg

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 for information on the structure of each response.

Table A-1: Reference Data Service Response Elements

Element	Description
responseError	Returned when a request cannot be completed for any reason. It is an errorInfo element.
securityData[ ]	Contains an array of securityData elements
securityData	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.
barData	Contains the response data for an IntradayBarRequest. It can provide a barTickData[ ] element and/or an eidData array element.
barTickData[ ]	Contains an array of barTickData elements
barTickData	Contains values associated to the bar, including time, open, high, low, close, volume, numEvents.
tickData	Contains the response data for an IntradayTickRequest. It can provide a tickData[ ] element and/or an eidData array element.
tickData[ ]	Contains an array of tickData elements
tickData[ ] :: tickData	Contains values associated to the eventType, including time, type, value, size, condition code, and exchange code.
eidData[ ]	Contains a list of eidData values associated to the securities requested. If the requestor does not have the entitlement as per EXCH<GO> then the identifiers will not be returned.
securityError	Returned when a request cannot be completed for any reason. It is an errorInfo element.
fieldExceptions[ ]	Contains an array of fieldExceptions.
fieldExceptions	Contains a field identifier, message and errorInfo element.
fieldData[ ]	Contains an array of fieldData values
fieldData	**Reference Data Request:** element with the fieldId and value **Historical Data Request:** element with the relativeDate, Date, fieldId and value
errorInfo	Contains values about the error which has occurred, including the source, code, category, message, and subcategory.

Bloomberg

Table A-2: Reference Data Service Response Values

Element	Type	Description
security	String	The security requested. See "Security/Securities" on page 76 for additional details..
eidData	Integer	Entitlement identifier (EID) associated to the requested security.
sequenceNumber	Integer	Security sequence number, specifying the position of the security in the request.
fieldId	String	Requested field represented as an alphanumeric or a Mnemonic, i.e. PR005 or PX_LAST.
relativeDate	String	Relative date string associated with this historical data-point. This field will only be returned if "returnRelativeDate" historical data request option is specified as "true".
Date	Date	Date associated with this historical data-point
Time	DateTime	Tick time for an intraday tick request
Type	String	The event type for an intraday tick
Value	Integer	Value of an eventType or field.
	Double	
	String	
	Date	
	Time	
	Datetime	
Size	Integer	Size of an event for intraday tick data (for example, number of shares).
conditionCode	String	A comma delimited list of exchange condition codes associated with the event.
exchangeCode	String	Single character indicating exchange tick event origin.
Source	String	Bloomberg internal error source information.
Code	Integer	Bloomberg internal error code.
Category	String	Bloomberg error classification. Used to determine the general classification of the failure.
message	String	Human readable description of the failure.
subcategory	String	(Optional) Bloomberg sub-error classification. Used to determine the specific classification of the failure.

Bloomberg

Table A-2: Reference Data Service Response Values

rpsCode	String	Transaction code.The following values appear: -B: A customer transaction where the dealer purchases securities from the customer. -S: A customer transaction where the dealersells securities to the customer. -D: An inter-dealer transaction (always from the sell side).
brokerBuyCode	String	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} <GO>**.
brokerSellCode	String	
micCode	String	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.

Bloomberg

A.3 Schema for API Field Service //blp//apiflds

A.3.1 Requests: Choice

Top level request to the service.

Element	Type	Description
fieldInfoRequest	FieldInfoRequest	Request for field information.
fieldSearchRequest	FieldSearchRequest	Field search information.
categorizedFieldSearchRequest	CategorizedFieldSearch Request	See "Categorized Field Search Request" on page 188.

A.3.2 Responses: Choice

Top level request to the service.

Element	Type	Description
fieldResponse	FieldResponse	Field response information.
categorizedFieldResponse	CategorizedFieldResponse	See "Categorized Field Search Request Response" on page 189.

A.3.3 Field Information Request

Identifier: the reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
id		string	See "Fields" on page 78 for additional details. Fields can be specified as a alpha numeric or mnemonic.
Example Syntax: `Element idList = request.GetElement("id");` `request.Append("id", "LAST_PRICE");` `request.Append("id", "pq005");`			
Return field documenation:			
Element	**Element Value**	**Type**	**Description**
returnFieldDocumentatio n	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.3.3.1 Field Information Request Response

See "Field Service Response Elements" on page 193 and "Field Service Response Values" on page 194 for more information.

Bloomberg

A.3.4 Field Search Request

Identifier: the reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
searchSpec		String	The string argument to search through mnemonics, descriptions and definitions. It is also able to 'intelligently' expand works, i.e. mkt ==> market.
Example Syntax: request.Set("searchSpec", "mutual fund");			
Include options:			
Element	**Element Value**	**Type**	**Description**
category	New Fields Analysis Corporate Actions Custom Fields Descriptive Earnings Estimates Fundamentals Market Activity Metadata Ratings Trading Systems	String	Categories for fields
productType	All	String	The results will be filtered by fields that are avaliable for this yellow key (security type).
	Govt	String	
	Corp	String	
	Mtge	String	
	M-Mkt	String	
	Muni	String	
	Pfd	String	
	Equity	String	
	Cmdty	String	
	Index	String	
	Curncy	String	

fieldType	All	String	Results include fields that are both streaming (real-time and delayed) and reference (static)
	Realtime	String	Results include fields that provide streaming data (real-time and delayed)
	Static	String	Results include fields that provide reference data (static).

```
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:

Element	Element Value	Type	Description
category	New Fields Analysis Corporate Actions Custom Fields Descriptive Earnings Estimates Fundamentals Market Activity Metadata Ratings Trading Systems	String	Categories for fields
productType	All	String	The results will be filtered by fields that are avaliable for this yellow key (security type).
	Govt	String	
	Corp	String	
	Mtge	String	
	M-Mkt	String	
	Muni	String	
	Pfd	String	
	Equity	String	
	Cmdty	String	
	Index	String	
	Curncy	String	

Bloomberg

fieldType	All	String	Results include fields that are both streaming (real-time and delayed) and reference (static)
	Realtime	String	Results include fields that provide streaming data (real-time and delayed)
	Static	String	Results include fields that provide reference data (static).

Example Syntax:	`Element element = request.getElement ("exclude");`
	`element.setElement("productType", "Equity");`
	`element.setElement("fieldType", "Static");`
	`Element element1 = element.GetElement("category");`
	`element1.AppendValue("Ratings");`
	`element1.AppendValue("Analysis");`

Return field documenation:			
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.3.4.1 Field Search Request Response

See "Field Service Response Elements" on page 193 and "Field Service Response Values" on page 194 for more information.

Bloomberg

Figure A-7: Field Search Request Response

A.3.5 Categorized Field Search Request

Identifier: the reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
searchSpec		String	The string argument to search through mnemonics, descriptions and definitions. It is also able to 'intelligently' expand works, i.e. mkt ==> market.
Example Syntax: `request.Set("searchSpec", "mutual fund");`			
Exclude options:			
Element	**Element Value**	**Type**	**Description**
category	New Fields Analysis Corporate Actions Custom Fields Descriptive Earnings Estimates Fundamentals Market Activity Metadata Ratings Trading Systems	String	Categories for fields
productType	All	String	The results will be filtered by fields that are avaliable for this yellow key (security type).
	Govt	String	
	Corp	String	
	Mtge	String	
	M-Mkt	String	
	Muni	String	
	Pfd	String	
	Equity	String	
	Cmdty	String	
	Index	String	
	Curncy	String	

fieldType	All	String	Results include fields that are both streaming (real-time and delayed) and reference (static)
	Realtime	String	Results include fields that provide streaming data (real-time and delayed)
	Static	String	Results include fields that provide reference data (static).

```
Example Syntax:  Element element = request.getElement ("exclude");
                 element.setElement("productType", "Equity");
                 element.setElement("fieldType", "Static");
                 Element element1 = element.GetElement("category");
                 element1.AppendValue("Ratings");
                 element1.AppendValue("Analysis");
```

Return field documenation:

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.3.5.1 Categorized Field Search Request Response

See <u>"Field Service Response Elements" on page 193</u> and <u>"Field Service Response Values" on page 194</u> for more information.

Bloomberg

CategorizedfieldResponse

The fieldResponse message has zero to one fieldSearchError elements

Categorizedfield SearchError
- source
- code
- category
- message
- Subcategory (optional)

category[ ]

category
- categoryName
- categoryId
- numFields
- description
- isLeafNode

fieldData[ ]

fieldData[ ] has zero to many fieldData elements

fieldData
- id

fieldData has zero to one fieldInfo

fieldInfo
- mnemonic
- datatype
- categoryName
- description
- documentation

property
- id
- value

fieldInfo has zero to one overrides[ ]

overrides[ ]
- override

fieldData has zero to one fieldError elements

fieldError
- source
- message
- code
- Subcategory (optional)
- category

A.3.6 Field List Request

Identifier: the reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
fieldType	All	String	Results include fields that are both streaming (real-time and delayed) and reference (static)
	Realtime	String	Results include fields that provide streaming data (real-time and delayed)
	Static	String	Results include fields that provide reference data (static).
`Example Syntax:   element.setElement("fieldType", "Static");`			
Return field documenation:			
Element	**Element Value**	**Type**	**Description**
returnFieldDocumentatio n	TRUE or FALSE	Boolean	Returns a description about the field as seen on FLDS<GO>. Default value is false.
`request.Set("returnFieldDocumentation", true);`			

A.3.6.1 Field List Request Response

See "Field Service Response Elements" on page 193 and "Field Service Response Values" on page 194 for more information.

Figure A-9: Field List Request Response

Bloomberg

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.

Element	Description
fieldSearchError	Returned when a request cannot be completed for any reason. It is an errorInfo element.
fieldData[ ]	Contains an array of fieldData values
fieldData	Contains a id corresponding to the requested field identifier, along with either a fieldInfo or fieldError element
fieldInfo	Contains values on the mnemonic, datatype, categoryName, description, and documentation.
fieldError	Returned when a request cannot be completed for any reason or in the case of a fieldInfoRequest when an invalid field mnemonic or alpha-numeric is entered.
categorizedFieldSearchError	Returned when a request cannot be completed for any reason. It is an errorInfo element.
category[ ]	Contains an array of category elements.
category	Contains categoryName, categoryId, numFields, descriptions, isLeafNode and a fieldData[ ] element.
errorInfo	Contains values about the error which has occurred, including the source, code, category, message, and subcategory.

Bloomberg

A.3.8 Field Service Response Values

Element	Type	Description
id	String	Resulting field represented as an alphanumeric or a Mnemonic, i.e., PR005 or PX_LAST.
mnemonic	Integer	Resulting field represented as a mnemonic, i.e., PX_LAST.
datatype	Enumeration	Enumeration values representing Bloomberg data types. Please see specific SDK documentation for the enum values.
ftype	Enumeration	Enumeration value representing data types shown in **XDM<GO>**.
categoryName	String	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.
description	String	Is the short description describing the field, for example for the mnemonic LAST_PRICE the description is "Last Trade/Last Price".
documentation	String	Corresponds to the definition in **FLDS<GO>**
Time	DateTime	Tick time for an intraday tick request
Type	String	The event type for an intraday tick
Source	String	Bloomberg internal error source information.
Code	Integer	Bloomberg internal error code.
Category	String	Bloomberg error classification. Used to determine the general classification of the failure.
message	String	Human readable description of the failure.
subcategory	String	Bloomberg sub-error classification. Used to determine the specific classification of the failure.

A.4 Market Bar Subscription

A.4.1 Market Bar Subscription Settings

Argument Value	Type	Description
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"`
field	string	The following fields are returned for Market Bars: TIME, OPEN, HIGH, LOW, CLOSE, NUMBER_OF_TICKS, VOLUME. These values are only updated on a trade. For this reason, LAST_PRICE should be submitted in the subscription string. See "Fields" on page 78 for additional details. Fields can be specified as a alpha numeric or mnemonic.
Example Syntax: `Subscription mySubscription = new Subscription("//blp/mktbar/ticker/VOD LN Equity", "LAST_PRICE", new CorrelationID(id));`		
interval	string	*Optional*. Interval time defined thelength in minutes of a bar. If undefined it is set to 1 minute. This is the minimum duration. The maximum duration is 1440 minutes, (=24 hours).
start_time	string	*Optional*. This should be in the format hh:mm. If these values are not specified then they default is time of subscription.
end_time	string	*Optional*. This should be in the format hh:mm. If these values are not specified then they default is session end time.
Example Syntax: ` Subscription mySubscription = new Subscription(security, field, "interval=5" "start_time=15:00", "end_time=15:30",CorrelationID(id));`		

A.4.2 Market Bar Subscription: Data Events Response

Argument Value	Type	Description
TIME	datetime	Returns the time of the last TRADE on every update.
Example Syntax: `Datetime time = msg.getElementAsDatetime(TIME);`		
OPEN	Float64	Returns open price for each bar. Will be returned in the first tick for the bar.
Example Syntax: `int open = msg.getElementAsFloat64(OPEN);`		
HIGH	Float64	Returns high price at the beginning of the bar and subsequently every higher price that occurs until the end of the bar.
Example Syntax: `int high = msg.getElementAsFloat64(HIGH);`		
LOW	Float64	Returns low price at the beginning of the bar and subsequently every higher price that occurs until the end of the bar.
Example Syntax: `int low = msg.getElementAsFloat64(LOW);`		

Bloomberg

Argument Value	Type	Description
CLOSE	Float64	Returns updated close price on every update.
`Example Syntax: int close = msg.getElementAsFloat64(CLOSE);`		
NUMBER_OF_TICKS	Int32	Counts tick number on every update until a new bar starts.
Example Syntax: `int number_of_ticks = msg.getElementAsInt32(NUMBER_OF_TICKS);`		
VOLUME	Int64	Volume increments for number of trades in each market bar and is reset at the start of each market bar.
Example Syntax: `float volume = msg.getElementAsInt64(VOLUME);`		

Bloomberg

A.5 Schema for Market Data and Custom VWAP

A.5.1 MarketDataEvents: Choice

Events related to Market Data:

Event Name	Type	Description
MarketDataEvents	MarketDataEvents	Market Data Eventss

A.5.2 Market Data Service Subscription Options

Argument Value	Type	Description
interval	string	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.

Example Syntax:
`Subscription mySubscription = new Subscription(security, fields,`
`                "interval=30.0", new CorrelationID(security));`		

delayed	string	Forces the subscription to be delayed even if the requestor has real-time exchange entitlements.

Example Syntax:
`Subscription mySubscription = new Subscription(security, fields,`
`                "delayed", new CorrelationID(security));`		

A.5.3 MarketDataEvents: Sequence

Fields in subscription

Element	Type	Description
TORONTO_MOC_ELIGIBLE_REALTIME	Optional Boolean	Toronto MOC Eligible
NASDAQ_CLOSING_CROSS_ELIGIBLE_RT	Optional Boolean	Nasdaq Closing Cross Eligible
MGF_SETTING_RT	Optional Boolean	MGF Setting (Real-time)
RT_EXCH_TRADE_STATUS	Optional Boolean	Exchange Trading Status
RT_QUOTE_STATUS	Optional Boolean	Quotation Status

Bloomberg

Element	Type	Description
IND_BID_FLAG	Optional Boolean	Indicative Bid Flag
IND_ASK_FLAG	Optional Boolean	Indicative Ask Flag
TRADING_DT_REALTIME	Optional Date	Trading Date
RT_TIME_OF_TRADE	Optional Datetime	Time Trade Occurred
CR_OBSERVATION_DATE	Optional Datetime	Current Observation Date
PRIOR_OBSERVATION_DATE	Optional Datetime	Prior Observation Date
TIME	Optional Datetime	Time of Last Update
VOLUME	Optional Int32	Volume
BID_YIELD	Optional Float32	Bid Yield
ASK_YIELD	Optional Float32	Ask Yield
RT_OPEN_INTEREST	Optional Float32	Open Interest (Real-time)
OFF_ON_EXCH_VOLUME_RT	Optional Int32	Off And On Exchange Volume (Real-time)
OFF_EXCH_VOLUME_RT	Optional Int32	Off Exchange Volume (Real-time)
PX_VOLUME_BAL_RT	Optional Int32	Volume Balance (Real-time)
OPT_DELTA_BID_RT	Optional Float32	Delta Bid (Real-time)
OPT_DELTA_ASK_RT	Optional Float32	Delta Ask (Real-time)
OPT_DELTA_MID_RT	Optional Float32	Delta Mid (Real-time)
OPT_DELTA_LAST_RT	Optional Float32	Delta Last Trade (Real-time)
OPT_GAMMA_BID_RT	Optional Float32	Gamma Bid (Real-time)
OPT_GAMMA_ASK_RT	Optional Float32	Gamma Ask (Real-time)
OPT_GAMMA_MID_RT	Optional Float32	Gamma Mid (Real-time)
OPT_GAMMA_LAST_RT	Optional Float32	Gamma Last Trade (Real-time)
OPT_VEGA_BID_RT	Optional Float32	Vega Bid (Real-time)
OPT_VEGA_ASK_RT	Optional Float32	Vega Ask (Real-time)
OPT_VEGA_MID_RT	Optional Float32	Vega Mid (Real-time)
OPT_VEGA_LAST_RT	Optional Float32	Vega Last Trade (Real-time)
OPT_IMPLIED_VOLATILITY_BID_RT	Optional Float32	Implied Volatility Bid (Real-time)
OPT_IMPLIED_VOLATILITY_ASK_RT	Optional Float32	Implied Volatility ASK (Real-time)
OPT_IMPLIED_VOLATILITY_MID_RT	Optional Float32	Implied Volatility Mid (Real-time)
OPT_IMPLIED_VOLATILITY_LAST_RT	Optional Float32	Implied Volatility Last Trade (Real-time)
EQY_SH_FOREIGN_RT	Optional Float32	Shares Available To Foreign Investors (Real-time)
LISTED_SH_RT	Optional Float32	Number Of Listed Shares (Real-time)

Bloomberg

Element	Type	Description
BLP_SPRD_TO_BENCH_BID_RT	Optional Float32	Bloomberg Bid Spread To Benchmark (Real-time)
BLP_SPRD_TO_BENCH_ASK_RT	Optional Float32	Bloomberg Ask Spread To Benchmark (Real-time)
BLP_SPRD_TO_BENCH_MID_RT	Optional Float32	Bloomberg Mid Spread To Benchmark (Real-time)
BLP_Z_SPRD_MID_RT	Optional Float32	Bloomberg Mid Z Spread (Real-time)
BLP_ASW_SPREAD_MID_RT	Optional Float32	Bloomberg Mid ASW Spread (Real-time)
BLP_I_SPRD_MID_RT	Optional Float32	Bloomberg Mid I Spread (Real-time)
BLP_CDS_BASIS_MID_RT	Optional Float32	Bloomberg Mid CDS Basis (Real-time)
BLP_SPRD_TO_BENCH_CHG_RT	Optional Float32	Bloomberg Sprd To Bench Chg On Day (Real-time)
BLP_Z_SPRD_CHG_RT	Optional Float32	Bloomberg Z Spread Change On Day (Real-time)
BLP_ASW_SPRD_CHG_RT	Optional Float32	Bloomberg ASW Spread Change On Day (Real-time)
BLP_I_SPRD_CHG_RT	Optional Float32	Bloomberg I Spread Change On Day (Real-time)
BLP_CDS_BASIS_CHG_RT	Optional Float32	Bloomberg CDS Basis Change On Day (Real-time)
BLP_SPRD_TO_BENCH_PCT_CHG_RT	Optional Float32	Bloomberg Spd To Bench % Chg On Day (Real-time)
BLP_Z_SPRD_PCT_CHG_RT	Optional Float32	Bloomberg Z Spread % Change On Day (Real-time)
BLP_ASW_SPRD_PCT_CHG_RT	Optional Float32	Bloomberg ASW Spread % Chg On Day (Real-time)
BLP_I_SPRD_PCT_CHG_RT	Optional Float32	Bloomberg I Spread % Change On Day (Real-time)
BLP_CDS_BASIS_PCT_CHG_RT	Optional Float32	Bloomberg CDS Basis % Change On Day (Real-time)
PX_SETTLE_ACTUAL_RT	Optional Float32	Settlement Price Actual (Real-time)
ARBITRAGE_ASK_ORD_NOT_MATCHED_RT	Optional Float32	Arbitrage Ask Orders Not Matched (Real-time)
ARBITRAGE_BID_ORD_NOT_MATCHED_RT	Optional Float32	Arbitrage Bid Orders Not Matched (Real-time)
NON_ARBITRAGE_ASK_NOT_MATCHED_RT	Optional Float32	Non Arbitrage Ask Orders Not Matched (Real-time)
NON_ARBITRAGE_BID_NOT_MATCHED_RT	Optional Float32	Non Arbitrage Bid Orders Not Matched (Real-time)
ARBITRAGE_ASK_ORD_VOLUME_RT	Optional Int32	Arbitrage Ask Orders Volume (Real-time)
ARBITRAGE_BID_ORD_VOLUME_RT	Optional Int32	Arbitrage Bid Orders Volume (Real-time)

Bloomberg

Element	Type	Description
NON_ARBIT_ASK_ORD_VOLUME_RT	Optional Int32	Non Arbitrage Ask Orders Volume (Real-time)
NON_ARBIT_BID_ORD_VOLUME_RT	Optional Int32	Non Arbitrage Bid Orders Volume (Real-time)
PRE_ANNOUNCE_NUM_PROG_ASK_RT	Optional Float32	Pre Announce Num of Program Ask Orders (Real-time)
PRE_ANNOUNCE_NUM_PROG_BID_RT	Optional Float32	Pre Announce Num of Program Bid Orders (Real-time)
TRUST_ASK_ORD_VOLUME_RT	Optional Int32	Trust Ask Orders Volume (Real-time)
PROPRIETARY_ASK_ORD_VOLUME_RT	Optional Int32	Proprietary Ask Orders Volume (Real-time)
TRUST_BID_ORD_VOLUME_RT	Optional Int32	Trust Bid Orders Volume (Real-time)
PROPRIETARY_BID_ORD_VOLUME_RT	Optional Int32	Proprietary Bid Orders Volume (Real-time)
TOTAL_VOLUME_PROGRAM_TRADE_RT	Optional Int32	Total Volume of Program Trading (Real-time)
PX_INDICATIVE_BID_SIZE_RT	Optional Int32	Indicative Bid Price Size (Real-time)
PX_INDICATIVE_ASK_SIZE_RT	Optional Int32	Indicative Ask Price Size (Real-time)
NUM_TRADES_RT	Optional Int32	Number Of Trades
MGF_VOLUME_RT	Optional Int32	MGF Volume (Real-time)
NUM_TRADES_OPENING_AUCTION_RT	Optional Int32	Number Of Trades In Opening Auction (Real-time)
NUM_TRADES_CLOSING_AUCTION_RT	Optional Int32	Number Of Trades In Closing Auction (Real-time)
ALL_PRICE_SIZE	Optional Int32	All Price Size
RT_NYSE_LIQUIDITY_BID_SIZE	Optional Int32	NYSE Liquidity Quote Bid Size
RT_NYSE_LIQUIDITY_ASK_SIZE	Optional Int32	NYSE Liquidity Quote Ask Size
VOLUME_THEO	Optional Int32	Theoretical Volume
SIZE_LAST_AT_TRADE	Optional Int32	Size of Last AT Trade
SIZE_LAST_AT_TRADE_TDY	Optional Int32	Size of Today's Last AT Trade
OPEN_YLD	Optional Float32	Open Yield
OPEN_YLD_TDY	Optional Float32	Today's Open Yield
HIGH_YLD	Optional Float32	High Yield
HIGH_YLD_TDY	Optional Float32	Today's High Yield
LOW_YLD	Optional Float32	Low Yield
LOW_YLD_TDY	Optional Float32	Today's Low Yield
LAST_YLD	Optional Float32	Last Yield
LAST_YLD_TDY	Optional Float32	Today's Last Yield
SIZE_LAST_TRADE_TDY	Optional Int32	Size of Today's Last Trade

Bloomberg

Element	Type	Description
LAST2_YLD	Optional Float32	Last 2 Yield
LAST_DIR_YLD	Optional Int32	Last Yield Direction
LAST2_DIR_YLD	Optional Int32	Second Last Yield Direction
PREV_SES_LAST_YLD	Optional Float32	Previous Session Last Yield
BID2_YLD	Optional Float32	Bid 2 Yield
ASK2_YLD	Optional Float32	Ask 2 Yield
BID_DIR_YLD	Optional Int32	Bid Yield Direction
ASK_DIR_YLD	Optional Int32	Ask Yield Direction
MID_DIR	Optional Int32	Mid Direction
MID2_DIR	Optional Int32	Second Mid Direction
RT_PX_CHG_PCT_1D	Optional Float32	Real-Time Price Change 1 Day Percent
RT_YLD_CHG_NET_1D	Optional Float32	Real-Time Yield Change 1 Day Net
RT_YLD_CHG_PCT_1D	Optional Float32	Real-Time Yield Change 1 Day Percent
ASK_SIZE_TDY	Optional Int32	Today's Ask Size
BID_SIZE_TDY	Optional Int32	Today's Bid Size
VOLUME_TDY	Optional Int32	Today's Volume
BID_YLD_TDY	Optional Float32	Today's Bid Yield
ASK_YLD_TDY	Optional Float32	Today's Ask Yield
UP_LIMIT	Optional Float32	Up Limit
DOWN_LIMIT	Optional Float32	Down Limit
LAST_DIR	Optional Int32	Last Direction
LAST2_DIR	Optional Int32	Second Last Direction
BID_DIR	Optional Int32	Bid Direction
ASK_DIR	Optional Int32	Ask Direction
SIZE_LAST_TRADE	Optional Int32	Size of Last Trade
ASK_SIZE	Optional Int32	Ask Size
BID_SIZE	Optional Int32	Bid Size
LAST_PRICE	Optional Float64	Last Price
BID	Optional Float64	Bid Price
ASK	Optional Float64	Ask Price
HIGH	Optional Float64	High Price
LOW	Optional Float64	Low Price
BEST_BID	Optional Float64	Best Bid
BEST_ASK	Optional Float64	Best Ask
MID	Optional Float64	Mid Price
LAST_TRADE	Optional Float64	Last Trade
OPEN	Optional Float64	Open Price

Bloomberg

Element	Type	Description
PREV_SES_LAST_PRICE	Optional Float64	Previous Session Last Price
EXCH_VWAP	Optional Float64	Exchange VWAP
NASDAQ_OPEN	Optional Float64	NASDAQ Official Open Price
NASDAQ_FIRST_TRADE	Optional Float64	NASDAQ First Actual Trade
NASDAQ_PREV_BID	Optional Float64	NASDAQ Prevailing Bid Price
NASDAQ_PREV_ASK	Optional Float64	NASDAQ Prevailing Ask Price
INDICATIVE_FAR	Optional Float64	Far Indicative Price
INDICATIVE_NEAR	Optional Float64	Near Indicative Price
IMBALANCE_BID	Optional Float64	Net Order Imbalance Bid Price
IMBALANCE_ASK	Optional Float64	Net Order Imbalance Ask Price
ORDER_IMB_BUY_VOLUME	Optional Int32	Net Order Imbalance Bid Volume
ORDER_IMB_SELL_VOLUME	Optional Int32	Net Order Imbalance Ask Volume
VWAP	Optional Float64	Eqty intraday VWAP
FIXING_RATE_REALTIME	Optional Float64	Fixing Rate
HIGH_TEMP_REALTIME	Optional Float64	High Temperature
LOW_TEMP_REALTIME	Optional Float64	Low Temperature
MEAN_TEMP_REALTIME	Optional Float64	Mean Temperature
HEATING_DAYS_REALTIME	Optional Float64	Heating Degree Days
COOLING_DAYS_REALTIME	Optional Float64	Cooling Degree Days
REL_HUMIDITY_REALTIME	Optional Float64	Relative Humidity
WIND_SPEED_REALTIME	Optional Float64	Wind Speed
WEATHER_CODE_REALTIME	Optional Float64	Weather Condition Code
PRECIPITATION_REALTIME	Optional Float64	Precipitation
MARKET_DEFINED_VWAP_REALTIME	Optional Float64	Market Defined VWAP (Real-time)
MIN_LIMIT	Optional Float64	Minimum Limit Price
MAX_LIMIT	Optional Float64	Maximum Limit Price
THEO_PRICE	Optional Float64	Theoretical Price
MIN_LIMIT_OUT_OF_SESSION	Optional Float64	Minimum Limit Price Out Of Session
MAX_LIMIT_OUT_OF_SESSION	Optional Float64	Maximum Limit Price Out Of Session
BID_WEIGHTED_AVG_SPREAD	Optional Float64	Bid Weighted Average Spread
ASK_WEIGHTED_AVG_SPREAD	Optional Float64	Ask Weighted Average Spread
RT_NYSE_LIQUIDITY_PX_BID	Optional Float64	NYSE Liquidity Quote Bid Price
RT_NYSE_LIQUIDITY_PX_ASK	Optional Float64	NYSE Liquidity Quote Ask Price
INDICATIVE_BID	Optional Float64	Indicative Bid Price
INDICATIVE_ASK	Optional Float64	Indicative Ask Price
PX_EVAL_JAPANESE_REALTIME	Optional Float64	Japanese Evaluation Price
LAST_ALL_SESSIONS	Optional Float64	Last Price All Sessions

Bloomberg

Element	Type	Description
PX_NASDAQ_VWOP_REALTIME	Optional Float64	NASDAQ VWOP Price
BLP_I_SPRD_LAST_RT	Optional Float64	Bloomberg Last I Spread (Real-time)
PREV_CLOSE_VALUE_REALTIME	Optional Float64	Previous Closing Value
BID_ALL_SESSION	Optional Float64	Bid Price All Session
ASK_ALL_SESSION	Optional Float64	Ask Price All Session
EBS_TOUCH_HIGH_REALTIME	Optional Float64	EBS Touch High
EBS_TOUCH_LOW_REALTIME	Optional Float64	EBS Touch Low
PX_PREV_TO_LAST_REALTIME	Optional Float64	Previous-To-Last Price
PX_TARGIN_SERVICE_REALTIME	Optional Float64	TARGIN Service Price (Real-time)
PX_TARGIN_OFFCIAL_REALTIME	Optional Float64	TARGIN Official Price (Real-time)
FOREIGN_HOLDING_PCT_RT	Optional Float64	Percentage Of Foreign Holding (Real-time)
OWNERSHIP_LIMIT_RATIO_RT	Optional Float64	Ownership Limit Ratio (Real-time)
RT_EVAL_JAPANESE_CHG_ON_DAY	Optional Float64	Japanese Evaluation Price Change On Day (Real-time)
RT_EVAL_JAPANESE_PCT_CHG_ON_DAY	Optional Float64	Japanese Eval Price Pct Change On Day (Real-time)
BLP_Z_SPRD_LAST_RT	Optional Float64	Bloomberg Last Z Spread (Real-time)
BLP_ASW_SPREAD_LAST_RT	Optional Float64	Bloomberg Last ASW Spread (Real-time)
BLP_RT_SPRD_TO_BENCH_LAST_RT	Optional Float64	Bloomberg Last Spread to Benchmark (Real-time)
TRUST_ASK_ORD_VALUE_RT	Optional Float64	Trust Ask Orders Value (Real-time)
PROPRIETARY_ASK_ORD_VALUE_RT	Optional Float64	Proprietary Ask Orders Value (Real-time)
TRUST_BID_ORD_VALUE_RT	Optional Float64	Trust Bid Orders Value (Real-time)
PROPRIETARY_BID_ORD_VALUE_RT	Optional Float64	Proprietary Bid Orders Value (Real-time)
TOTAL_VALUE_PROGRAM_TRADE_RT	Optional Float64	Total Value of Program Trading (Real-time)
PX_OFFICIAL_AUCTION_RT	Optional Float64	Official Auction Price (Real-time)
NYSE_LRP_HIGH_PRICE_RT	Optional Float64	NYSE LRP High Price (Real-time)
NYSE_LRP_LOW_PRICE_RT	Optional Float64	NYSE LRP Low Price (Real-time)
ALL_PRICE	Optional Float64	All Price
BEST_BID1	Optional Float64	Best Bid 1
BEST_BID2	Optional Float64	Best Bid 2
BEST_BID3	Optional Float64	Best Bid 3
BEST_BID4	Optional Float64	Best Bid 4
BEST_BID5	Optional Float64	Best Bid 5
BEST_ASK1	Optional Float64	Best Ask 1
BEST_ASK2	Optional Float64	Best Ask 2

Bloomberg

Element	Type	Description
BEST_ASK3	Optional Float64	Best Ask 3
BEST_ASK4	Optional Float64	Best Ask 4
BEST_ASK5	Optional Float64	Best Ask 5
BEST_BID1_SZ	Optional Int32	Best Bid 1 Size
BEST_BID2_SZ	Optional Int32	Best Bid 2 Size
BEST_BID3_SZ	Optional Int32	Best Bid 3 Size
BEST_BID4_SZ	Optional Int32	Best Bid 4 Size
BEST_BID5_SZ	Optional Int32	Best Bid 5 Size
BEST_ASK1_SZ	Optional Int32	Best Ask 1 Size
BEST_ASK2_SZ	Optional Int32	Best Ask 2 Size
BEST_ASK3_SZ	Optional Int32	Best Ask 3 Size
BEST_ASK4_SZ	Optional Int32	Best Ask 4 Size
BEST_ASK5_SZ	Optional Int32	Best Ask 5 Size
LAST_AT_TRADE	Optional Float64	Last AT Trade
LAST2_AT_TRADE	Optional Float64	Last 2 AT Trade
LAST_AT_TRADE_TDY	Optional Float64	Today's Last AT Trade
MID_TDY	Optional Float64	Today's Mid Price
MID2	Optional Float64	Mid 2 Price
RT_PX_CHG_NET_1D	Optional Float64	Real-Time Price Change 1 Day Net
OPEN_TDY	Optional Float64	Today's Open Price
LAST_PRICE_TDY	Optional Float64	Today's Last Price
BID_TDY	Optional Float64	Today's Bid Price
ASK_TDY	Optional Float64	Today's Ask Price
HIGH_TDY	Optional Float64	Today's High Price
LOW_TDY	Optional Float64	Today's Low Price
LAST2_PRICE	Optional Float64	Last 2 Price
BID2	Optional Float64	Bid 2 Price
ASK2	Optional Float64	Ask 2 Price
RT_EXCH_MARKET_STATUS	Optional String	Exchange Market Status
RT_TRADING_PERIOD	Optional String	Trading Period
BID_BROKER_CODE	Optional String	Bid Broker Code
ASK_BROKER_CODE	Optional String	Ask Broker Code
IMBALANCE_INDIC_RT	Optional String	Imbalance Indicator
BLP_SPREAD_BENCHMARK_NAME_RT	Optional String	Bloomberg Spread Benchmark Name (Real-time)
BLP_SWAP_CURVE_NAME_RT	Optional String	Bloomberg Swap Curve Name (Real-time)

Bloomberg

Element	Type	Description
FINANCIAL_STATUS_INDICATOR_RT	Optional String	Financial Status Indicator (Real-time)
BID_YLD_COND_CODE	Optional String	Bid Yield Condition Code
YLD_COND_CODE	Optional String	Yield Condition Code
ASK_YLD_COND_CODE	Optional String	Ask Yield Condition Code
ALL_PRICE_COND_CODE	Optional String	
BID_COND_CODE	Optional String	Bid Condition Codes
ASK_COND_CODE	Optional String	Ask Condition Codes
RT_SIMP_SEC_STATUS	Optional String	Simplified Security Status
RT_PRICING_SOURCE	Optional String	Real-Time Pricing Source
NYSE_LRP_SEND_TIME_RT	Optional Time	NYSE LRP Send Time (Real-time)
BID_ASK_TIME	Optional Time	Time of Last Bid/Ask Update
SES_START	Optional Time	Session Start
SES_END	Optional Time	Session End
TRADE_SPREAD_TIME	Optional Time	Time of TRADE_SPREAD tick
NEWS_STORY_TIME	Optional Time	Time of NEWS_STORY tick
BID_TIME	Optional Time	Time of BID tick
BID_BEST_TIME	Optional Time	Time of BID_BEST tick
VOLUME_UPDATE_TIME	Optional Time	Time of VOLUME_UPDATE tick
MARKET_DEPTH_TIME	Optional Time	Time of MARKET_DEPTH tick
CANCEL_CORRECT_TIME	Optional Time	Time of CANCEL_CORRECT tick
MIN_LIMIT_OUT_OF_SESSION_TIME	Optional Time	Time of MIN_LIMIT_OUT_OF_SESSION tick
BID_SPREAD_TIME	Optional Time	Time of BID_SPREAD tick
BT_MKT_TURN_TIME	Optional Time	Time of BT_MKT_TURN tick
HIGH_TIME	Optional Time	Time of HIGH tick
BT_LSE_LAST_TIME	Optional Time	Time of BT_LSE_LAST tick
AT_TRADE_TIME	Optional Time	Time of AT_TRADE tick
ASK_YEILD_TIME	Optional Time	Time of ASK_YEILD tick
PRICE_UPDATE_TIME	Optional Time	Time of PRICE_UPDATE tick
OPEN_INTEREST_TIME	Optional Time	Time of OPEN_INTEREST tick
VOLUME_TIME	Optional Time	Time of VOLUME tick
EVAL_JAPANESE_TIME	Optional Time	Time of EVAL_JAPANESE tick
ASK_WEIGHTED_AVG_SPREAD_TIME	Optional Time	Time of ASK_WEIGHTED_AVG_SPREAD tick
THEO_PRICE_TIME	Optional Time	Time of THEO_PRICE tick
BUY_SELL_INFO_TIME	Optional Time	Time of BUY_SELL_INFO tick
SETS_MID_PRICE_TIME	Optional Time	Time of SETS_MID_PRICE tick

Bloomberg

Element	Type	Description
TAKE_TIME	Optional Time	Time of TAKE tick
TICK_NUM_TIME	Optional Time	Time of TICK_NUM tick
SMART_TIME	Optional Time	Time of SMART tick
INDICATIVE_ASK_TIME	Optional Time	Time of INDICATIVE_ASK tick
BT_SEC_ASK_TIME	Optional Time	Time of BT_SEC_ASK tick
LOW_TIME	Optional Time	Time of LOW tick
BT_SEC_BID_TIME	Optional Time	Time of BT_SEC_BID tick
LOW_YIELD_TIME	Optional Time	Time of LOW_YIELD tick
MAX_LIMIT_TIME	Optional Time	Time of MAX_LIMIT tick
TRADING_PERIOD_TIME	Optional Time	Time of TRADING_PERIOD tick
INDICATIVE_BID_TIME	Optional Time	Time of INDICATIVE_BID tick
API_INTERNAL_TIME	Optional Time	Time of API_INTERNAL tick
ASK_LIFT_TIME	Optional Time	Time of ASK_LIFT tick
NYSE_LIQUIDITY_ASK_TIME	Optional Time	Time of NYSE_LIQUIDITY_ASK tick
BID_YEILD_TIME	Optional Time	Time of BID_YEILD tick
ASK_BEST_TIME	Optional Time	Time of ASK_BEST tick
MKT_INDICATOR_TIME	Optional Time	Time of MKT_INDICATOR tick
NYSE_LIQUIDITY_BID_TIME	Optional Time	Time of NYSE_LIQUIDITY_BID tick
SMART_QUOTE_TIME	Optional Time	Time of SMART_QUOTE tick
NEW_MKT_DAY_TIME	Optional Time	Time of NEW_MKT_DAY tick
MAN_TRADE_WITH_SIZE_TIME	Optional Time	Time of MAN_TRADE_WITH_SIZE tick
BT_ASK_RECAP_TIME	Optional Time	Time of BT_ASK_RECAP tick
BT_MID_PRICE_TIME	Optional Time	Time of BT_MID_PRICE tick
BID_MKT_MAKER_TIME	Optional Time	Time of BID_MKT_MAKER tick
SETTLE_TIME	Optional Time	Time of SETTLE tick
HIT_TIME	Optional Time	Time of HIT tick
BT_LAST_RECAP_TIME	Optional Time	Time of BT_LAST_RECAP tick
LAST_TRADE_TIME	Optional Time	Time of LAST_TRADE
PRE_POST_MARKET_TIME	Optional Time	Time of PRE_POST_MARKET tick
ALL_PRICE_TIME	Optional Time	Time of ALL_PRICE tick
OPEN_TIME	Optional Time	Time of OPEN tick
HIGH_YIELD_TIME	Optional Time	Time of HIGH_YIELD tick
ASK_MKT_MAKER_TIME	Optional Time	Time of ASK_MKT_MAKER tick
MAX_LIMIT_OUT_OF_SESSION_TIME	Optional Time	Time of MAX_LIMIT_OUT_OF_SESSION tick
SMARTMAX_TIME	Optional Time	Time of SMARTMAX tick
YIELD_TIME	Optional Time	Time of YIELD tick

Bloomberg

Element	Type	Description
VWAP_TIME	Optional Time	Time of VWAP tick
BID_WEIGHTED_AVG_SPREAD_TIME	Optional Time	Time of BID_WEIGHTED_AVG_SPREAD tick
ASK_TIME	Optional Time	Time of ASK tick
MIN_LIMIT_TIME	Optional Time	Time of MIN_LIMIT tick
ASK_SPREAD_TIME	Optional Time	Time of ASK_SPREAD tick
SETTLE_YIELD_TIME	Optional Time	Time of SETTLE_YIELD tick
BID_LIFT_TIME	Optional Time	Time of BID_LIFT tick
BT_BID_RECAP_TIME	Optional Time	Time of BT_BID_RECAP tick

Bloomberg

A.5.4 Market VWAP Service Subscription Options

Argument Value	Type	Description
VWAP_START_TIME	string	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<GO>** timing for Desktop API and UTC for Server API.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_START_TIME=11:00",```
``` new CorrelationID(security));```		

VWAP_END_TIME	string	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<GO>** timing for Desktop API and UTC for Server API.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_END_TIME=12:00",```
``` new CorrelationID(security));```		

VWAP_MIN_SIZE	string	Minimum trade volume for a trade to be included in VWAP computation. Values are taken as signed integers.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_MIN_SIZE=1000",```
``` new CorrelationID(security));```		

VWAP_MAX_SIZE	string	Maximum trade volume for a trade to be included in VWAP computation. Values are taken as signed integers.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_MAX_SIZE=2000",```
``` new CorrelationID(security));```		

VWAP_MIN_PX	string	Minimum trade price for a trade to be included in VWAP computation. Values are taken as floats.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_MIN_PX=23.5",```
``` new CorrelationID(security));```		

VWAP_MAX_PX	string	Maximum trade price for a trade to be included in VWAP computation. Values are taken as floats.

Example Syntax:
```Subscription mySubscription = new Subscription(topic + security,```
```   fields, "&VWAP_MAX_PX=25.5",```
``` new CorrelationID(security));```		

A.6 Schema for API Authorization

Element	Description
AuthorizationRequest	Requests Bloomberg to check if a given Bloomberg Anywhere user is logged into the BLOOMBERG PROFESSIONAL service at a specified location.
UserAsidEquivalenceRequest	*Deprecated.* Compares the exchanges entitlements of a given user to the exchange entitlements of the ServerAPI.
LogonStatusRequest	Requests a user's logon status for their Bloomberg Anywhere.
UserEntitlementsRequest	Requests a list of the user's exchange entitlements
SecurityEntitlementsRequest	Requests a list of a specific security's exchange entitlements
SecurityEntitlementsByUserRequest	*Deprecated.* Requests a list of exchange entitlements for a security by user.
TokenRequest	*Deprecated.* Requests a token.

A.6.1 Authorization Request

Bloomberg UUID: the Bloomberg unique user identifier			
Element	**Element Value**	**Type**	**Description**
uuid		integer	The Bloomberg unique user identifier
Example Syntax: `Request request = authSvc.CreateAuthorizationRequest();` `request.Set("uuid", 11223344);`			
IP Address: Location of where the user is viewing the ServerAPI data			
Element	**Element Value**	**Type**	**Description**
ipAddress		string	

Example Syntax:
 `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.			
Element	**Element Value**	**Type**	**Description**
requireAsidEquivalence	TRUE or FALSE	Boolean	When set to 'true', the AuthorizationRequest will succeed only if the users permission are equal to or greater than that of the Server API.
Example Syntax: request.Set("requireAsidEquivalence", true);			
Token: *Deprecated*. Authorizes the user with the token based approach.			
Element	**Element Value**	**Type**	**Description**
token			Token returned by TokenRequest for a user. (Optional. Either ipAddress or token must be supplied.)

A.6.2 Authorization Request Response

See "Field Service Response Elements" on page 215 and "Field Service Response Elements" on page 215.

A.6.3 Logon Status Request

Bloomberg UUID: the Bloomberg unique user identifier			
Element	**Element Value**	**Type**	**Description**
uuid		integer	The Bloomberg Unique User Identifier (UUID)
sid			Deprecated. do not use
sidInstance			Deprecated. do not use
terminalSid			Deprecated. do not use
terminalSidInstance			Deprecated. do not use.

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			
Element	**Element Value**	**Type**	**Description**
ipAddress		string	The location where the user is viewing API data

Example Syntax:
```Request logonStatusRequest = authSvc.CreateRequest("LogonStatusRequest");```
```logonStatusRequest.Set("ipAddress", "111.22.33.44");```			

A.6.4 Logon Status Request Response

See and .

A.6.5 User Entitlements Request

Bloomberg UUID: the Bloomberg unique user identifier			
Element	**Element Value**	**Type**	**Description**
uuid		integer	The Bloomberg Unique User Identifier (UUID)
sid			*Deprecated*. do not use
sidInstance			*Deprecated*. do not use
terminalSid			*Deprecated*. do not use
terminalSidInstance			*Deprecated*. do not use.

Example Syntax:
 	
`Request request = authSvc.CreateRequest("UserEntitlementsRequest");`
`Element userinfo = request.GetElement("userInfo");`
`userinfo.SetElement("uuid", 11223344);`			

A.6.6 User Entitlements Request Response

See "Field Service Response Elements" on page 215 and "Field Service Response Elements" on page 215.

A.6.7 Security Entitlements Request

Securities: the reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
securities		string	Element holding the list of securities to retrieve exchange entitlements.

Example Syntax:
`Request request = authSvc.CreateRequest("SecurityEntitlementsRequest");`
`Element securities = request.GetElement("securities");`
`securities.AppendValue("IBM US Equity");`			

A.6.8 Security Entitlements Request Response

See and .

A.6.9 Authorization Token Request

Identifier: The Bloomberg Unique User Identifier.			
Element	**Element Value**	**Type**	**Description**
uuid		integer	The Bloomberg Unique User Identifier (UUID)

Example Syntax:
`Request request = authSvc.CreateRequest("AuthorizationTokenRequest");`
`request.Set("uuid", 11223344);`			

Label: A label that identifies which Server API application is requesting the token.			
Element	**Element Value**	**Type**	**Description**
label		string	String identifier for the requesting ServerAPI application

Example Syntax:
`Request request = authSvc.CreateRequest("AuthorizationTokenRequest");`
`request.Set("label", "myApp");`			

A.6.10 Authorization Token Request Response

See and .

Bloomberg

A.6.11 Field Service Response Elements

Element	Description
AuthorizationSuccess	Returned for an authorization request when the UUID provided is logged into the Bloomberg Anywhere at the specified IP address.
AuthorizationFailure	Returned for an authorization request on failure. It is an errorInfo element.
reason	An AuthorizationFailure message will contain one "reason" element
responseError	Returned when a request cannot be completed for any reason. It is an errorInfo element.
errorInfo	Contains values about the error which has occurred, including the source, code, category, message, and subcategory.
eidData[ ]	Contains a list of eidData elements, each associated to a security requested.
eidData[ ]::eidData	Contains status, sequence number and list of entitlement identifiers.
eids[ ]	Contains a list of entitlementId values associated to the user.

A.6.12 Field Service Request Values

Element	Type	Description
Source	String	Bloomberg internal error source information.
Code	Integer	Bloomberg internal error code.
Category	String	Bloomberg error classification. Used to determine the general classification of the failure.
message	String	Human readable description of the failure.
subcategory	String	(Optional) Bloomberg sub-error classification. Used to determine the specific classification of the failure.
entitlementId	Integer	Entitlement identifier (EID)
status	Integer	Status where success = 0. Any other code indicates failure.
sequenceNumber	Integer	Security sequence number, specifying the position of the security in the request.
isLoggedOn	Boolean	Returns true when the UUID specified in logged into the BLOOMBERG PROFESSIONAL service at the specified IP address.

Bloomberg

B Java Examples

This section contains the following code examples and sample output from each example:

B.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
          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);
         }
      }
   }
}
```

B.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
            }
        }
    }
}
```

B.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);
            }
        }
    }
}
```

Bloomberg

Subscription Paradigm Output

```
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
    }
```

B.3 Asynchronous Event Handling

```java
// 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 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();
            }
    }
}
```

```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;
          }
```

```
            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;
            }
         }
      }
    }
  }
```

B.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
```

Bloomberg

B.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));
```

Bloomberg

```java
        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.getElementAsInt32("sequenceNumber");
                if (securityData.hasElement("securityError")) {
                    Element securityError =
                                  securityData.getElement("securityError");
                    System.out.println("* security       =" + security);
                    //Element securityError = securityData.getElement(
                                                        "securityError");
                    securityError.print(System.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.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 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);
            }
        }
    }
}
```

B.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      =BLAHBLAHBLAH US Equity
securityError = {
    source = 193::bbdbs1
    code = 15
    category = BAD_SEC
    message = Unknown/Invalid security [nid:193]
    subcategory = INVALID_SECURITY
}
```

Bloomberg

B.5 Subscription Multiple

```java
// 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("MSFTT 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();
           }
    }
  }
```

B.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
```

Bloomberg

```
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_LAST_PRICE = 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
    }
handleDataEvent: 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
        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 = -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
        LAST2_PRICE = 92.51
        LAST_DIR = -1
        LAST2_DIR = 1
        BID_DIR = -1
        ASK_DIR = 1
        BID2 = 92.56
        ASK2 = 92.61
        ASK_SIZE = 5
        BID_SIZE = 1
        TIME = 21:15:12.000+00:00
        API_MACHINE = p065
        TRADE_SIZE_ALL_SESSIONS_RT = 500
        EID = 14003
        IS_DELAYED_STREAM = false
    }
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_SES_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
```

B.6 Authorization by IP Address

```java
// 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;
    }
```

![Bloomberg]

```java
    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);
            }
        }
    }
```

```java
    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);
        }
```

Bloomberg

```java
        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;
            }
        }
    }
}
```

Bloomberg

C .Net Examples

This section contains the following code examples:

Bloomberg

C.1 RequestResponseParadigm

```
// RequestResponseParadigm.cs

using System;
using System.Collections.Generic;
using System.Text;

using CorrelationID  = Bloomberglp.Blpapi.CorrelationID;
using Element        = Bloomberglp.Blpapi.Element;
using Event          = Bloomberglp.Blpapi.Event;
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 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.EventType.SESSION_STATUS == eventObj.Type
                && message.MessageType.Equals("SessionTerminated"))
                {
                    System.Console.WriteLine("Terminating: " +
                                            message.MessageType);
                    System.Environment.Exit(1);
                }
            }
        }
    }
}
```

Bloomberg

C.1.1 Request Response Paradigm Output

```
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
            }
        }
    }
}
```

C.2 Subscription Paradigm

```
// SubscriptionParadigm.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 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);
                }
            }
        }
    }
}
```

Bloomberg

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
```

Bloomberg

```
BEST_BID = 90.88
BEST_ASK = 90.9
LAST_TRADE = 90.89
VWAP = 91.6348
LAST_ALL_SESSIONS = 90.89
BID_ALL_SESSION = 90.88
ASK_ALL_SESSION = 90.9
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
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
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
```

Bloomberg

```
        TIME = 18:45:45.000+00:00
        TRADE_SIZE_ALL_SESSIONS_RT = 100
        EID = 14005
        IS_DELAYED_STREAM = false
    }
```

Bloomberg

C.3 Asynchronous Event Handling

```
// 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=" + eventObj.Type);
            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.Console.Error.WriteLine(
                                     "Could not open //blp/refdata for
async");
                                    System.Environment.Exit(1);
                                }
                            }
                            else
                            {
                                System.Console.Error.WriteLine(
                                        "Could not start session.");
                                System.Environment.Exit(1);
                            }
                         }
                        break;
                    }
```

Bloomberg

```csharp
                case Event.EventType.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;
                    }
            }
        }
    }
}
```

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 = {
                PX_LAST = 91.85
            }
        }
    }
}
```

C.4 Request Response Multiple

```csharp
// RequestResponseMultiple.cs

using System;
using System.Collections.Generic;
using System.Text;

using CorrelationID  = Bloomberglp.Blpapi.CorrelationID;
using Element        = Bloomberglp.Blpapi.Element;
using Event          = Bloomberglp.Blpapi.Event;
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 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");
                                                        // Last Price
            request.GetElement("fields").AppendValue("DS002");
                                                        // Description
            request.GetElement("fields").AppendValue("VWAP_VOLUME");
             // Volume used to calculate the Volume Weighted Average Price
            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;
                }
```

Bloomberg

```
                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);
            }
        }
    }
}
}
```

C.4.1 Request Response Multiple: Output

```
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      =BLAHBLAHBLAH US Equity
securityError = {
    source = 236::bbdbs2
    code = 15
    category = BAD_SEC
    message = Unknown/Invalid security [nid:236]
    subcategory = INVALID_SECURITY
}
```

Bloomberg

C.5 Subscription Multiple

```
// SubscriptionMultiple.cs

using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

using CorrelationID  = Bloomberglp.Blpapi.CorrelationID;
using Event          = Bloomberglp.Blpapi.Event;
using EventHandler   = Bloomberglp.Blpapi.EventHandler;
using Message        = Bloomberglp.Blpapi.Message;
using Session        = Bloomberglp.Blpapi.Session;
using SessionOptions = Bloomberglp.Blpapi.SessionOptions;
using Subscription   = Bloomberglp.Blpapi.Subscription;

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(
                                        new SubscriptionEventHandler(
                                            "myLabel",
                                    System.Console.Out).ProcessEvent));
```

```
        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);
        }
      }
    }
  }
```

C.5.1 Multiple Subscription: Output

```
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 = {
}
```

Bloomberg

```
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
    }
```

D C++ Examples

This section contains the following code examples:

-
-
-
-
-

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.

Bloomberg

D.1 RequestResponseParadigm

```cpp
// 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 <string.h>  // 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

```
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
             }
          }
    }
```

D.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;
    }
```

Bloomberg

Subscription Paradigm Output

```
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
    PUT_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
  }
```

D.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()
{
}
```

Bloomberg

```cpp
    // MANIPULATORS
    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 << "Unxepected 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;
}
```

Bloomberg

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
```

D.4 Request Response Multiple

```cpp
// RequestResponseParadigm.cpp

#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;
```

```
            } else {
                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 ouput 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);
        }
    }
}
```

Bloomberg

```cpp
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 calcuate 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
          case Event::PARTIAL_RESPONSE:
            handleResponseEvent(event);
            break;
          default:
            handleOtherEvent(event);
            break;
        }
    }
```

```
    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      =BLAHBLAHBLAH US Equity
securityError =  {
    source = 119::bbdbs1
    code = 15
    category = BAD_SEC
    message = Unknown/Invalid security [nid:119]
    subcategory = INVALID_SECURITY
 }
```

D.5 Subscription Multiple

```cpp
// 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);
};
```

```cpp
// 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
```

Bloomberg

```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;
}
```

Bloomberg

Subscription Multiple: Output

```
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
      LAST_DIR = 1
      LAST2_DIR = 1
      BID_DIR = 1
      ASK_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
 }
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
```

Bloomberg

```
        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
    }
```

Bloomberg

E C Examples

This section contains the following code examples:

-
-
-
-
-

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.

Bloomberg

E.1 RequestResponseParadigm

```c
/* 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);
}
```

Bloomberg

```c
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);
```

Bloomberg

```
      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;
  }
```

Bloomberg

Request Response Paradigm Output

```
EventType=2
correlationId=0 0 0
messageType=SessionStarted
SessionStarted =  {
 }
EventType=9
correlationId=0 0 0
messageType=ServiceOpened
ServiceOpened =  {
 }
Event Type = 5
correlationId=1 0 1
messageType  =ReferenceDataResponse
ReferenceDataResponse =  {
    securityData[] =
        securityData =  {
            security = IBM US Equity
            eidData[] =

            fieldExceptions[] =

            sequenceNumber = 0
            fieldData =  {
                PX_LAST = 91.170000
             }
         }
 }
```

Bloomberg

E.2 Subscription Paradigm

```c
/* 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);
}
```

Bloomberg

```c
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 //blp/mktdata.\n");
        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);
```

E C Examples   309

```
    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;
}
```

Bloomberg

Subscription Paradigm Output

```
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
updateCount = 3
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
  }
```

E.3 Asynchronous Event Handling

```c
/* 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.value.intValue);
```

```
        messageElements = blpapi_Message_elements(message);
        assert(messageElements);
        blpapi_Element_print(messageElements, &streamWriter, stdout, 0, 4);
    }
}

#ifdef __cplusplus
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);
                exit(1);
            }
        }
        break;
```

Bloomberg

```
        }
    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;
        }
        case BLPAPI_EVENTTYPE_PARTIAL_RESPONSE: {
          dumpEvent(event);
          break;
        }
        case BLPAPI_EVENTTYPE_RESPONSE: {
          dumpEvent(event);
          assert(session);
          printf("terminate process from handler\n");
          blpapi_Session_stop(session);
          exit(0);
          break;
        }
        default: {
          fprintf(stderr, "default-case\n");
          fprintf(stderr, "Unxepected Event Type %d\n",
                          blpapi_Event_eventType(event));
          exit(1);
          break;
        }
      }
    }
```

```c
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
```

E.4 Request Response Multiple

```c
/* 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)) {
```

```
            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      =%s\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);
```

Bloomberg

```c
        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);
```

Bloomberg

```
    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;
}
```

Bloomberg

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
 }
```

Bloomberg

E.5 Subscription Multiple

```
/* 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);
```

```
    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   = "MSFTT 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");
```

Bloomberg

```c
        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;
    }
```

Bloomberg

Subscription Multiple Output

```
handleStatusEventHandler: enter
handler label=myLabel
eventType=2
messageType=SessionStarted
correlationId=0 0 0
SessionStarted =  {
 }
handleStatusEventHandler: leave
handleStatusEventHandler: enter
handler label=myLabel
eventType=9
messageType=ServiceOpened
correlationId=0 0 0
ServiceOpened =  {
 }
handleStatusEventHandler: leave
handleStatusEventHandler: enter
handler 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
handler 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
    }
```