OPEN API

Services & schemas reference guide

Version: 1.4.1
Last Updated: 11/27/2015

Bloomberg
FOR ENTERPRISE

Related Documents

DOCUMENT NAME
Core User Guide
Core Developer's Guide
Enterprise User Guide
Enterprise Developer Guide
Publishing User Guide
Publishing Developer Guide

Bloomberg
FOR ENTERPRISE

Contents

Bloomberg
FOR ENTERPRISE

1 About This Guide

The Reference Guide will form the basis for understanding of the Bloomberg schemas and services.

1.1 SCHEMA

The role of the schema is to define the format of requests to the service, as well as the Events returned from that service. Within a service, one or more Event types may exist, each having its own schema. The schema is the shape of the data. For instance, market data is flat, while reference data is nested (like XML).

Each of the following sections provides an overview of the Request options and response structures 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.

☞ *For additional information, refer to the "Core User Guide".*

1.2 SERVICES

Schemas act to define the format of Requests to a service as well as the Events returned from that service. Within a service, one or more Event types may exist, each having its own schema. The schema is the shape of the data. For instance, market data is flat, while reference data is nested (like XML). Additional services have been created solely for B-PIPE users; these are covered in the B-PIPE training course. They are a by-product of the BPOD (B-PIPE-On-Demand) and Broadcast B-PIPE products, which are being replaced by the B-PIPE product. These include MSG1 Message Scraping (`//blp/msgscrape`), Full Market Depth (`//blp/mktdepthdata`), Market List (`//blp/mktlist`) and Source Reference (`//blp/srcref`).

☞ *For additional information refer to the '**Core User Guide**'.*

Bloomberg FOR ENTERPRISE

2 Schema Data Types

SEQUENCE
- Used to indicate an array of results, either values or structures.

INT32 AND INT64
- 32-bit and 64-bit signed integers

STRING
- Text data of undefined size

FLOAT32 AND FLOAT64
- 32-bit and 64-bit floating point value

ENUMERATION
- Enumerated value. The API provides a list of constants to use for each enumerator. Examples are language, periodicity, screentype, etc.

BOOL
- Boolean value, “true” or “false”; used for flags or switches.

DATETIME
- UTC date and time values, for example, “2011-09-03T01:04:04.000+01:00”

The role of the schema is to define the format of Requests to the service, as well as the Events returned from that service. Within a service, one or more Event types may exist, each having its own schema. The schema is the shape of the data. For instance, market data is flat (all fields at top level), while reference data is nested (XML).

Each element possesses the following properties and attributes:

- **Name**: The name of the Element.
- **Status**: ACTIVE — Available or INACTIVE — Unavailable
- **Type**: Data type of that Element. It includes SEQUENCE (group), ENUMERATION, BOOL, STRING, etc.
- **Minimal Occurrence**: 0 — Optional or 1 — Required
- **Maximal Occurrence**: 1 — Element or -1 — Array

DESCRIPTION	MINIMAL OCCURRENCE	MAXIMAL OCCURRENCE
Optional Field	0	1
Required Field	1	1
Array	1	-1

2.1 SCHEMA SAMPLE

Shown below is a sample that provides details of the schema.

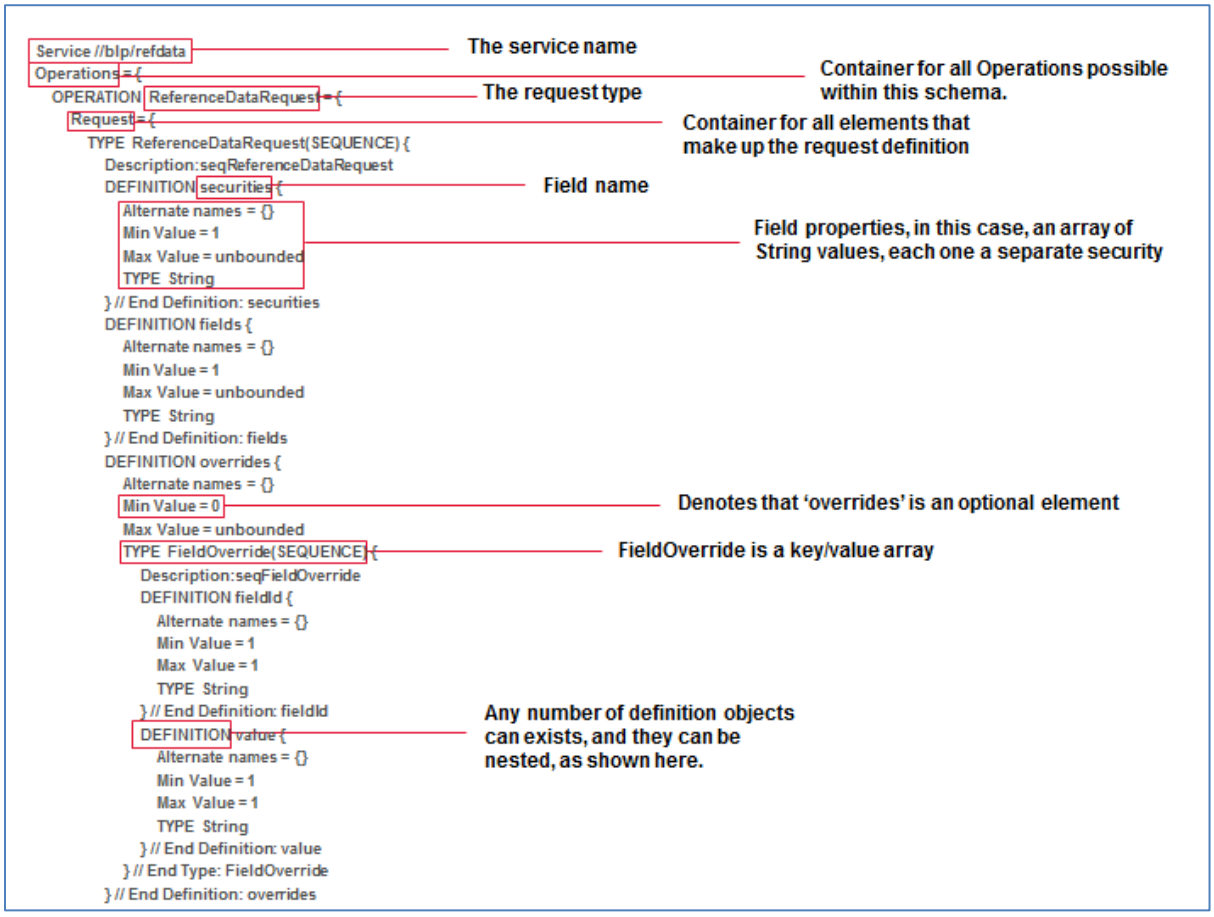


Figure 1. Schema Sample

☞ *For additional information, refer to the "Core User Guide".*

Bloomberg FOR ENTERPRISE

3 List of Services

Every service has its own schema, each explained in detail, not that not all services are available for all products:

- **`//blp/mktdata`** — The service streaming for Market Data
- **`//blp/refdata`** — The service for static Reference Data
- **`//blp/vwap`** — The service for Volume-Weighted Average Price
- **`//blp/apiflds`** — The service for API Field Service
- **`//blp/instruments`** — The service for Security Lookup
- **`//blp/mktbar`** — The service for real-time and delayed Intraday Bars
- **`//blp/mktdepthdata`** — The service for Market-Depth Data
- **`//blp/mktlist`** — The service for Security Topic Lists
- **`//blp/srcref`** — The service for Source Reference
- **`//blp/apiauth`** — The service for Authorization and Permissioning

4 Security Nomenclature

Most services allow subscribing or requesting instruments using various nomenclatures. This, for example, allows asking for the same security by its "Yellow Key", Bloomberg Open Symbology, or an independent identifier service's reference id.

☞ *For additional information, refer to the "Core User Guide".*

/cusip [a]	Requests by CUSIP
/sedol [a]	Requests by SEDOL
/isin [a]	Requests by ISIN
/bsid [b]	Requests by Bloomberg Security Identifier
/bsym [a]	For requests by Bloomberg Security Symbol
/buid [a]	For requests by Bloomberg Unique Identifier
/eid [b]	For requests by Entitlement ID
/source [c]	For requests by Source syntax
/gdco [d]	For Requests by GDCO syntax
/bpkbl [a]	Requests by Bloomberg Parsekeyable Identifier
/ticker [b]	Requests by Bloomberg Ticker
/bbgid [a]	Requests by Bloomberg Global Identifier

[a] *topic types consist of source and the value of a given identifier separated by the forward slash: <source>/<identifier>*
[b] *topic types do not require a source and consist of value alone <Identity>*
[c] *topic type consists of only a <source>*
[d] *topic type consists of Broker ID and Mon ID separated by the forward slash: <broker_id>/<mon_id>*

EXAMPLES:

IDENTIFIER	API
Parsekyable:	//blp/mktdata/ticker/IBM UN Equity
BBGID:	//blp/mktdata/bbgid/BBG000BLNQ16
ISIN:	//blp/mktdata/isin/US4592001014 UN
CUSIP:	//blp/mktdata/cusip/459200101 UN
SEDOL:	//blp/mktdata/sedol/2005973
BSYM:	//blp/mktdata/bsym/UN/IBM

5 Streaming Market Data (//blp/mktdata)

The market data service ("`//blp/mktdata`") enables retrieval of streaming data for securities that 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 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 (Initial Paint) Message, will contain a value for all the available fields specified in the Subscription. Subsequent Messages may contain values for some or all of the requested Bloomberg fields. A Message might contain none of the requested Bloomberg fields — Messages are only filtered based on the fields they *could* contain rather than the fields they actually contain, with many fields in the streaming Events being optional. The Bloomberg API will ensure that all Messages containing any of the fields explicitly subscribed to will be pushed to the application. Finally, the stream may return additional fields, not included in the Subscription, in these Messages. These additional fields are not filtered for the purpose of speed, and their inclusion is subject to change at any time. Please note that B-PIPE users do have the option to enable field filtering, which will result in only the fields subscribed being returned. For simplicity, this course will assume that field filtering is not applied.

The following example shows how to subscribe for streaming data.

```
<C++>

// Assume that session already exists and the "//blp/mktdata" service
has

// been successfully opened.

SubscriptionList subscriptions;

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);
```

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 can be used to populate their own caches. However, at times these fields will be set to a null value. For BID and ASK fields, this is usually interpreted as an instruction to

clear the values in the caches. It is important to test to see if the field is null before trying to retrieve a value from it.

5.1 MARKET DATA EVENT TYPES AND SUB-TYPES

A Subscription-based application is expected to handle a number of possible market data Event types and sub-types. When subscribing to market data for a security, the API:

Retrieves and delivers a summary of the current state of the security. A summary consists of data elements known as “fields”. The set of summary fields varies depending on the asset class of the requested security.

Streams all market data updates as they occur until Subscription cancellation. About 300 market data fields are available via the API Subscription interface, most of them derived from trade and quote Events.

An Event of type SUBSCRIPTION_DATA will contain a MessageType of “MarketDataEvents”, which contains any of the following market data Event types (e.g., MKTDATA_EVENT_TYPE):

SUMMARY
This market data Event type Message can be any of the following market Event sub-types (MKTDATA_EVENT_SUBTYPE):

INITPAINT — Message is the Initial Paint (a.k.a. snapshot), which is the most recent value for all the fields specified in the Subscription, as well as possibly other fields not included in the Subscription. The inclusion of these extra fields is done to enhance performance on the Bloomberg. If the Subscription is interval-based (i.e., an interval of n > 0), only SUMMARY INITPAINT Messages will be received every n number of seconds as the header is basically sent at the interval points with the latest tick values.

INTRADAY — Message indicates a regular summary Message, which is usually sent near the beginning of a zero-interval-based Subscription (closely after the INITPAINT SUMMARY Messages). It is an update to the snapshot (INITPAINT) Message.

NEWDAY — Sent from the Bloomberg Data Center to indicate that a new market day has occurred for the particular instrument subscribed to. It is sent after the market has closed and before the market opens the next day. Many times the first occurrence of this tick will be received an hour or two after the market close. More than one such tick can be received between the market close and market open. This is the time where certain fields are re-initialized to zero, such as VOLUME (total number of securities traded that day), to prepare for the new day.

INTERVAL — Returned only when making an interval-based Subscription. All messages will be of this type/sub-type. An INITPAINT message or any QUOTE or TRADE type Messages will not be received.

DATALOSS — Indicates that data has been lost. The Library drops Events when the number of Events outstanding for delivery exceeds the specified threshold controlled by SessionOptions.maxEventQueueSize. The correlationID property attached to the DATALOSS Message identifies the affected Subscription.

TRADE

This market data Event type indicates that this Event contains a trade Message and can be any of the following market data sub-types (MKTDATA_EVENT_SUBTYPE):

- **NEW** — Message contains a regular trade tick.
- **CANCEL** — Message contains cancellation of a trade.
- **CORRECTION** — Message contains correction to a trade.

QUOTE

This market data Event type Message can be of any one of the following market Event sub-types (MKTDATA_EVENT_SUBTYPE):

- **BID** — Single BID type field inside along with its applicable value.
- **ASK** — Single ASK type field inside along with its applicable value.
- **MID** — Single MID type field inside along with its applicable value.
- **PAIRED** — Both single ASK and BID type fields inside along with their applicable values (available only for the B-PIPE product).

6 Static Reference Data (//blp/refdata)

The reference data service provides the ability to access the following Bloomberg data with the Request/Response paradigm:

- **Reference Data**: Provides a snapshot of the current value of a security/field pair.
- **Historical End-of-Day Data**: Provides end-of-day data over a defined period of time for a security/field pair.
- **Historical Intraday Tick Data**: Provides each tick over a defined period of time for a single security and one or more Event types.
- **Historical Intraday Bar Data**: Provides a series of intraday summaries over a defined period of time for a single security and Event type.
  - ☞ ***Note**: Although other types of data are available under the `//blp/refdata` service, the aforementioned types are the most common and will serve as the primary focus of this module.*
- **Note**: Only the ReferenceDataRequest type is available for NONBPS users, and only for a subset of fields, on the reference data service. All other Request types on the reference data service are not supported.

6.1 OPERATIONS

OPERATION NAME	REQUEST TYPE	RESPONSE TYPE	DESCRIPTION
HistoricalData	HistoricalDataRequest	HistoricalDataResponse	**Request Historical Data**
IntraDayTick	IntraDayTickRequest	IntraDayTickResponse	**Request Intraday Tick Data**
IntraDayBar	IntraDayBarRequest	IntradayBarResponse	**Request Intraday Bar Data**
ReferenceData	ReferenceDataRequest	ReferenceDataResponse	**Request Reference Data**
PortfolioData	PortfolioDataRequest	PortfolioDataResponse	**Request Portfolio Data**
BeqsRequest	BeqsRequest	BeqsResponse	**Request EQS Screen Data**

6.2 REFERENCEDATAREQUEST: SEQUENCE

Securities: **A stock or bond**			
Element	**Element**	**Type**	Description
securities	string array	string	**Array of securities to fetch corresponding fields**
Example Syntax: `Element securities = request.GetElement("securities");` `securities.AppendValue("VOD LN Equity");`			
Fields: **The reference fields desired that correspond to data points. See FLDS <GO> for more information.**			
Element	**Element**	**Type**	Description

fields		string	
Example Syntax: `Element fields = request.GetElement("fields"); fields.AppendValue("PX_LAST");`			
Overrides: **Append overrides to modify the calculation**			
Element	**Element**	**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**	**Type**	Description
returnEids	TRUE or FALSE	Boolean	**Setting to true populates fieldData with an extra element containing a name and value for the EID date.**
Example Syntax: `request.Set("returnEids", true);`			
Return Formatted Value: R**eturns all data as a data type string**			
Element	**Element**	**Type**	Description
returnFormattedValue	TRUE or FALSE	Boolean	**Setting to true forces 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**	**Type**	Description
useUTCTime	TRUE or FALSE	Boolean	**Setting to true returns values in UTC. Setting to false causes default to the TZDF <GO> settings of the requestor.**
Example Syntax: `request.Set("useUTCTime", true);`			
Forced Delay: R**eturns latest reference data up to delay period**			
Element	**Element**	**Type**	Description
forcedDelay	TRUE or FALSE	Boolean	**Setting to true returns the latest data up to the delay period specified by the Exchange for this security. For example, requesting VOD LN Equity and PX_LAST returns a snapshot of the last price from 15 mins ago.**
Example Syntax: `request.Set("forcedDelay", true);`			

6.3 REFERENCEDATARESPONSE: CHOICE

The figure below shows the structure of a ReferenceDataResponse.

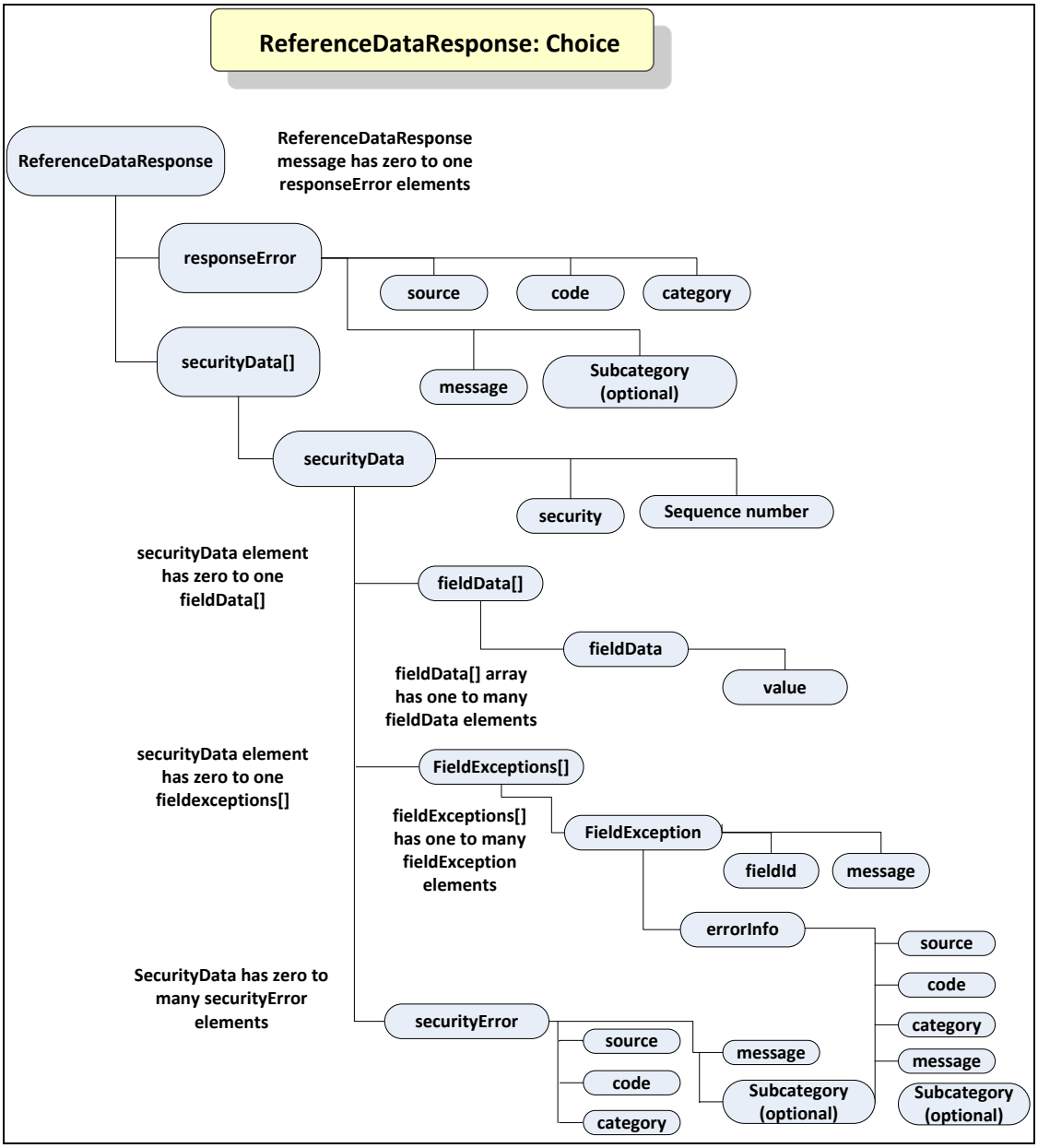


Figure 2. Structure of a ReferenceDataResponse

6.4 HISTORICALDATAREQUEST: SEQUENCE

Securities: A stock or bond			
Element	**Element Value**	**Type**	**Description**
securities		string	Array of securities to fetch corresponding fields
Example Syntax: `Element securities = request.GetElement("securities");` `securities.AppendValue("VOD LN Equity");`			
Fields: Reference fields desired that correspond to data points. See FLDS <GO> for more information.			
Element	**Element Value**	**Type**	**Description**
fields		string array	
Example Syntax: `Element fields = request.GetElement("fields");` `fields.AppendValue("PX_LAST");`			
Start Date: First date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
startDate	yyyymmdd	string	Start date in a year/month/day format
Example Syntax: `request.Set("startDate", "20090601");`			
End Date: End date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
endDate	yyyymmdd	string	End date in a year/month/day format. Will default to the current day if not specified.
Example Syntax: `request.Set("endDate", "20100601");`			
Period Adjustment: Determines 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");`			

Period Selection: Determines 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 BloombergProfessional 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 closing price in quote calculation.
	OVERRIDE_OPTION_GPA	string	Use 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 data is 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" 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	Response contains up to X data points, where X is the integer specified. If the original data set is larger than X, the response is 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 populates fieldData with an extra element containing a name and value for 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 populates 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);`			

Adjustment Abnormal: Adjusts 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 follows the DPDF <GO> BloombergProfessional service function. True is 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 2-character calendar code null terminated string. This will cause the data to be aligned according to the calendar and include calendar holidays. 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 2-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.
calendarOverrides 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.

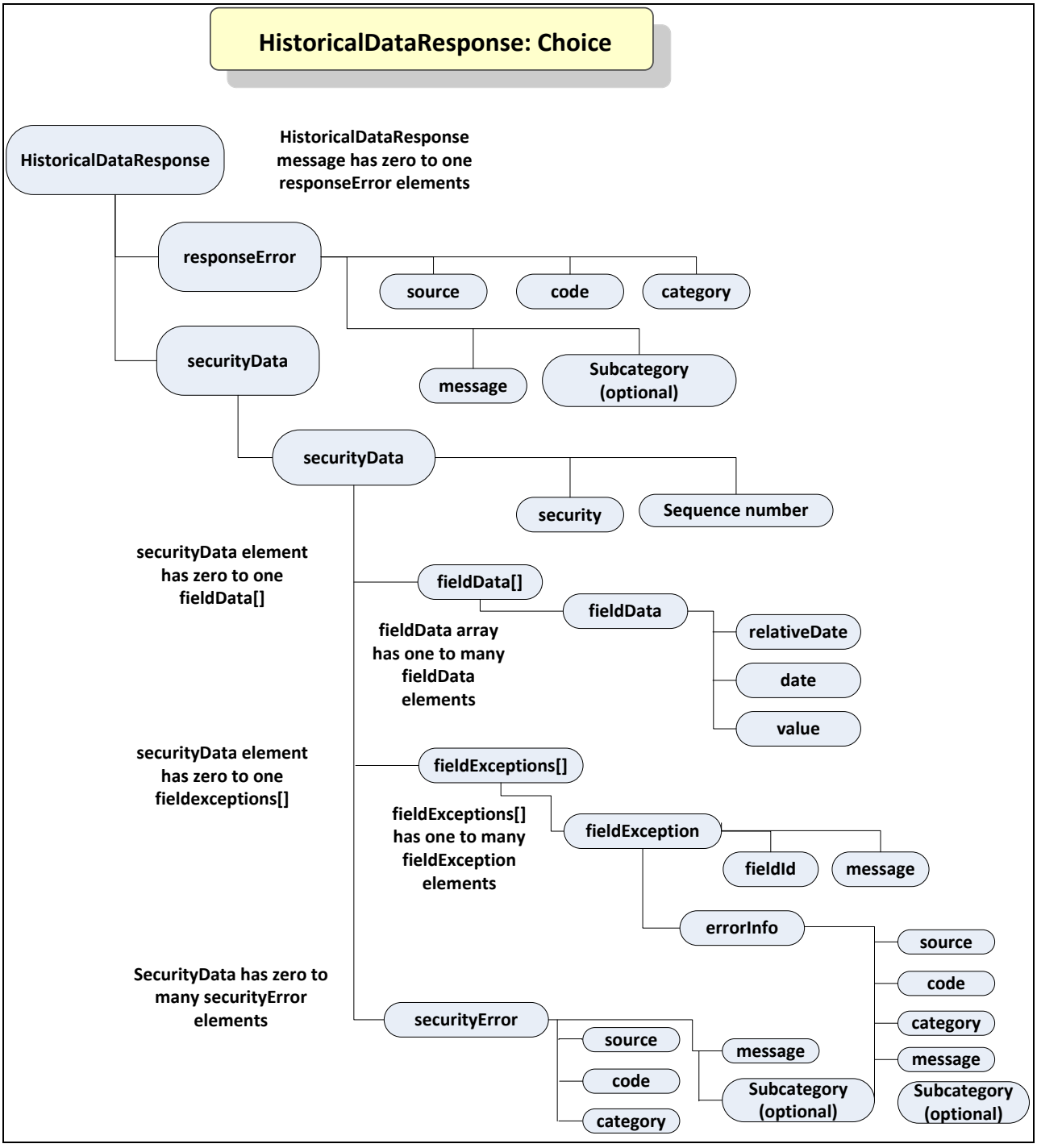
Element	Element Value	Type	Description
fieldID		string	Specify a field mnemonic or alpha- numeric, such as PR092 or PRICING_SOURCE. Review FLDS <GO> for list of possible overrides.
value		string	The desired override value

Example Syntax:

```
Element overrides = request["overrides"]; Element
override1 = overrides.AppendElement();
override1.SetElement("fieldId", "BEST_DATA_SOURCE_OVERRIDE");
override1.SetElement("value", "BLI");
```

6.5 HISTORICALDATARESPONSE: CHOICE

The figure below shows the structure of a Historical Data Response



6.6 INTRADAYTICKREQUEST: SEQUENCE

Securities: A stock or bond			
Element	**Element Value**	**Type**	**Description**
securities		string	Array of securities to fetch corresponding fields
Example Syntax: `Element securities = request.GetElement("securities");` `request.Set("security", "VOD LN Equity");`			
Start Date: 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: 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: Requested data Event type			
Element	**Element Value**	**Type**	**Description**
eventType	TRADE	string	Corresponds to LAST_PRICE
	BID	string	Depending on the Exchange, bid ticks returned as BID, BID_BEST or BEST_BID.
	ASK	string	Depending on the Exchange, ask ticks returned as ASK, ASK_BEST or BEST_ASK.
	BID_BEST	string	Depending on the Exchange, bid ticks returned as BID, BID_BEST or BEST_BID.
	ASK_BEST	string	Depending on the Exchange, ask ticks 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 returned as BID, BID_BEST or BEST_BID.
	BEST_ASK	string	Depending on the Exchange, ask ticks returned as ASK, ASK_BEST or BEST_ASK.
Example Syntax: `request.Set("eventType", "TRADE");`			

Include Condition Codes: Returns 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: Returns ticks in the Responses 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: Returns the Exchange code of the trade			
Element	**Element Value**	**Type**	**Description**
includeExchangeCodes	TRUE or FALSE	Boolean	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: Returns broker code of the trade.			
Element	**Element Value**	**Type**	**Description**
includeBrokerCodes	TRUE or FALSE	Boolean	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: Returns transaction codes.			
Element	**Element Value**	**Type**	**Description**

includeRpsCodes	TRUE or FALSE	Boolean	The reporting party side. The following values appear: -B: Customer transaction where dealer purchases securities from customer -S: Customer transaction where the dealer sells securities to the customer -D: Inter-dealer transaction (always from the sell side)
Example Syntax: `request.Set("includeRpsCodes", true);`			
Include Bank/Market Identifier Codes: Returns 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; it indicates venue on which trade was executed.
Example Syntax: `request.Set("includeBicMicCodes", true);`			

6.7 INTRADAYTICKRESPONSE: CHOICE

The figure below shows the structure of an IntradayTickResponse.

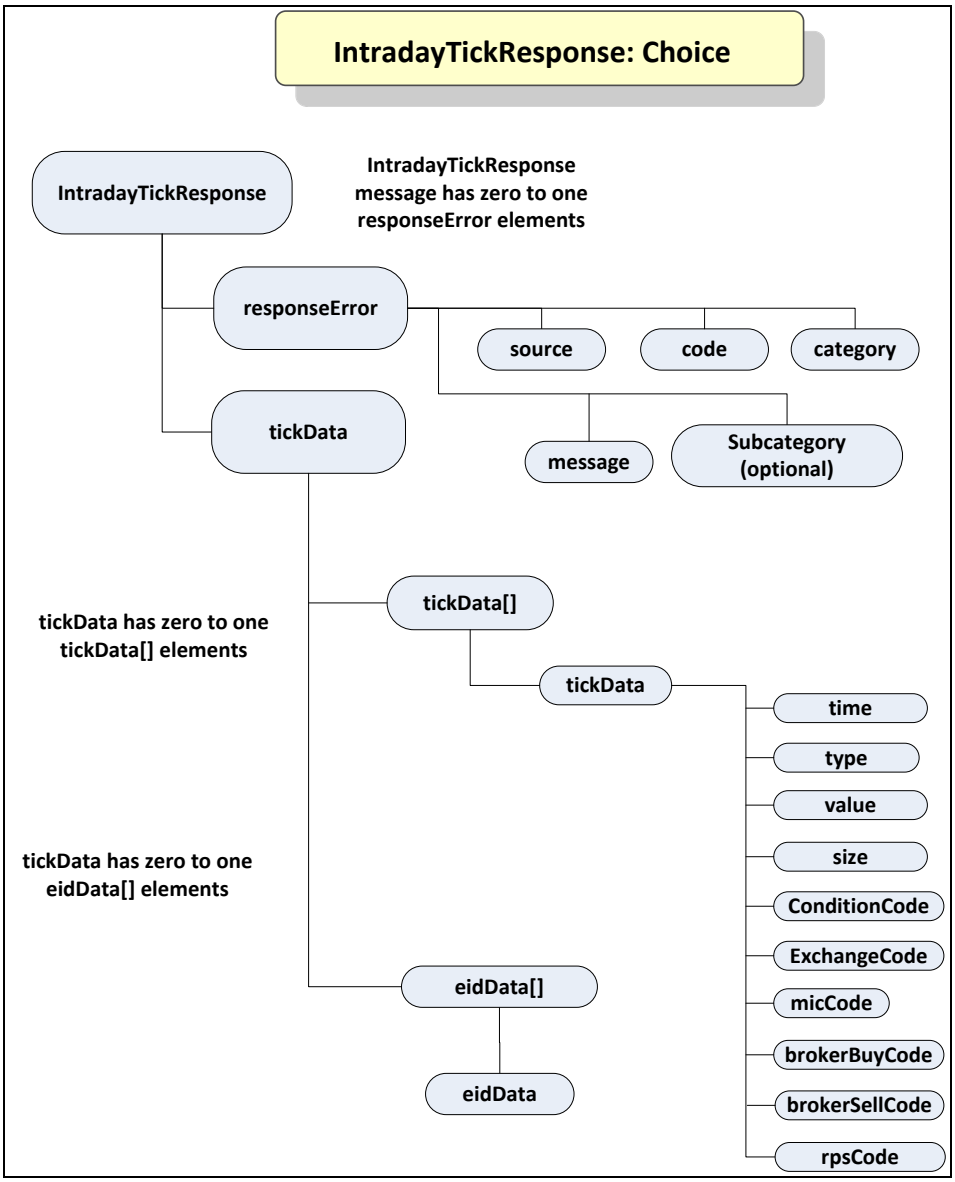


Figure 6. Intraday TickResponse

6.8 INTRADAYBARREQUEST: SEQUENCE

Securities: A stock or bond			
Element	**Element Value**	**Type**	**Description**
security		string	Array of securities to fetch corresponding fields
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	Start date and time
Example Syntax: `request.Set("startDateTime", "2010-04-27T15:55:00");`			
End Date: End date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
endDateTime	yyyy-mm-dd Thh:mm:ss	string	End date and time
Example Syntax: `request.Set("endDateTime", "2010-04-27T16:00:00");`			
Event Type: Requested data Event type			
Element	**Element Value**	**Type**	**Description**
eventType	TRADE	string	Corresponds to LAST_PRICE
	BID	string	Depending on the Exchange, bid ticks returned as BID, BID_BEST or BEST_BID.
	ASK	string	Depending on the Exchange, ask ticks returned as ASK, ASK_BEST or BEST_ASK.
	BID_BEST	string	Depending on the Exchange, bid ticks returned as BID, BID_BEST or BEST_BID.
	ASK_BEST	string	Depending on the Exchange, ask ticks returned as ASK, ASK_BEST or BEST_ASK.
	BEST_BID	string	Depending on the Exchange, bid ticksreturned as BID, BID_BEST or BEST_BID.
	BEST_ASK	string	Depending on the Exchange, ask ticks returned as ASK, ASK_BEST or BEST_ASK.
Example Syntax: `request.Set("eventType", "TRADE");`			
Interval: 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 1,440 in minutes. If omitted, the Request will default to 1 minute. One minute is the lowest possible granularity.
Example Syntax: `request.Set("interval", 60);`			
Gap Fill Initial Bar: Populate an empty bar with previous value			
Element	**Element Value**	**Type**	**Description**
gapFillInitialBar	TRUE or FALSE	Boolean	When set to true, a bar contains the previous bar values if there was no tick during this time interval.
Example Syntax: `request.Set("gapFillInitialBar", true);`			
Return Entitlements: Returns the entitlement identifiers associated with security.			
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);`			

6.8.1 STARTDATEINTERVAL

Start Date			
Element	**Element Value**	**Type**	**Description**
startDateTime	yyyy-mm-dd Thh:mm:ss	datetime	First date of the period to retrieve data
Example Syntax: `request.Set("startDateTime", "2010-04-27T9:30:00");`			
End Date			
Element	**Element Value**	**Type**	**Description**
endDateTime	yyyy-mm-dd Thh:mm:ss	datetime	End date of the period to retrieve data
Example Syntax: `request.Set("endDateTime", "2010-04-28T15:55:00");`			

6.8.2 STARTDATERANGEDURATION

Securities: A stock or bond			
Element	**Element Value**	**Type**	**Description**
rangeStartDateTimeList		datetime	maxOccurs = "unbounded"
Example Syntax: `Element securities = request.GetElement("securities");` `request.Set("security", "VOD LN Equity");`			
Start Date: First date of the period to retrieve data			
Element	**Element Value**	**Type**	**Description**
Duration	yyyy-mm-dd Thh:mm:ss	Int32	Start date and time
Example Syntax: `request.Set("startDateTime", "2010-04-27T15:55:00");`			

6.8.3 DATETIMEINFO CHOICE

Security			
Element	**Element Value**	**Type**	**Description**
security		string	seqIntradayBarDateTimeChoiceRequest
Example Syntax:			
Event Type			
Element	**Element Value**	**Type**	**Description**
eventType		BarEventType	
Example Syntax:			
Interval			
Element	**Element Value**	**Type**	**Description**
Interval		Int32	
Example Syntax:			
DateTimeInfo			
Element	**Element Value**	**Type**	**Description**
dateTimeInfo		dateTimeInfo	Choice of setting start end datetime or list start datetimes and duration
Example Syntax:			
GapFillInitialBar			
Element	**Element Value**	**Type**	**Description**
gapFillInitialBar		Boolean	minOccurs="0", maxOccurs="1"
Example Syntax:			
ReturnEIDs			
Element	**Element Value**	**Type**	**Description**
returnEids		Boolean	minOccurs="0", maxOccurs="1"
Example Syntax:			
AdjustmentNormal			
Element	**Element Value**	**Type**	**Description**
adjustmentNormal		Boolean	minOccurs="0", maxOccurs="1" alternateName>CshAdjNormal
Example Syntax:			

6.8.4 INTRADAYBARDATETIMECHOICEREQUEST: SEQUENCE

Security			
Element	**Element Value**	**Type**	**Description**
security		string	seqIntradayBarDateTimeChoiceRequest
Example Syntax:			
Event Type			
Element	**Element Value**	**Type**	**Description**
eventType		BarEventType	
Example Syntax:			
Interval			
Element	**Element Value**	**Type**	**Description**
Interval		integer	Sets the length of each time bar in the Response. Entered as a whole number, between 1 and 1,440 in minutes. If omitted, the request will default to 1 minute.
Example Syntax:			
DateTimeInfo			
Element	**Element Value**	**Type**	**Description**
dateTimeInfo		DateTimeInfo	Choice of setting start end datetime or list start datetimes and duration
Example Syntax:			
GapFillInitialBar			
Element	**Element Value**	**Type**	**Description**
gapFillInitialBar		Boolean	minOccurs="0", maxOccurs="1"
Example Syntax:			
ReturnEIDs			
Element	**Element Value**	**Type**	**Description**
returnEids		Boolean	minOccurs="0", maxOccurs="1"
Example Syntax:			
AdjustmentNormal			
Element	**Element Value**	**Type**	**Description**
adjustmentNormal		Boolean	minOccurs="0", maxOccurs="1" alternateName>CshAdjNormal
Example Syntax:			

AdjustmentAbnormal			
Element	**Element Value**	**Type**	**Description**
adjustmentAbnormal		Boolean	alternateName>CshAdjAbnormal
Example Syntax:			
AdjustmentSplit			
Element	**Element Value**	**Type**	**Description**
adjustmentSplit		Boolean	alternateName>CapChg
Example Syntax:			
adjustmentFollowDPDF			
Element	**Element Value**	**Type**	**Description**
adjustmentFollowDPDF		Boolean	alternateName>UseDPDF
Example Syntax: `request.Set("adjustmentNormal", true);`			
MaxdataPoints			
Element	**Element Value**	**Type**	**Description**
maxDataPoints		Int32	
Example Syntax: `request.Set("adjustmentAbnormal", true);`			
ForcedDelay			
Element	**Element Value**	**Type**	**Description**
forcedDelay		Boolean	
Example Syntax:			

6.8.5 INTRADAYBARRESPONSE: CHOICE

The figure below shows the structure of an IntradayBarResponse.

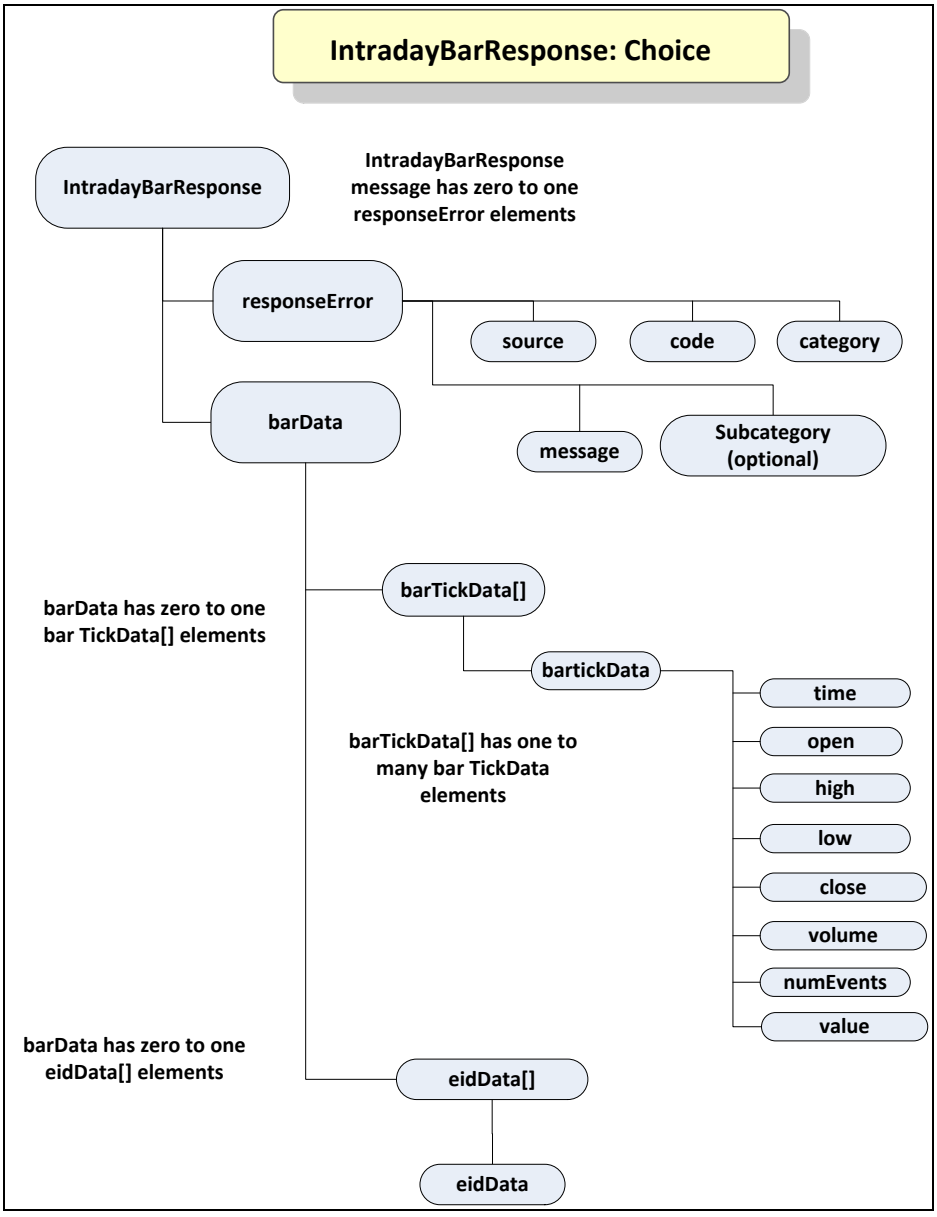


Figure 7. IntradayBarResponse

6.9 PORTFOLIODATAREQUEST: SEQUENCE

Securities: Portfolio ID			
Element	**Element Value**	**Type**	**Description**
securities	string array	string	The user's portfolio is identified by its Portfolio ID, which can be found on the upper right-hand corner of the settings tab on the portfolio's PRTU <GO> page on the Bloomberg Professional service.
Example Syntax: `Element securities = request.GetElement("securities"); securities.AppendValue("UXXXXXXX-X Client");`			
Fields: 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: 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	Date in "yyyymmdd" format
Example Syntax: `Element overrides = request["overrides"]; Element override1 = overrides.AppendElement(); override1.SetElement("fieldId", "REFERENCE_DATE"); override1.SetElement("value", "20100111");`			

6.10 PORTFOLIODATARESPONSE: CHOICE

The figure below shows the structure of a PortfolioDataResponse.

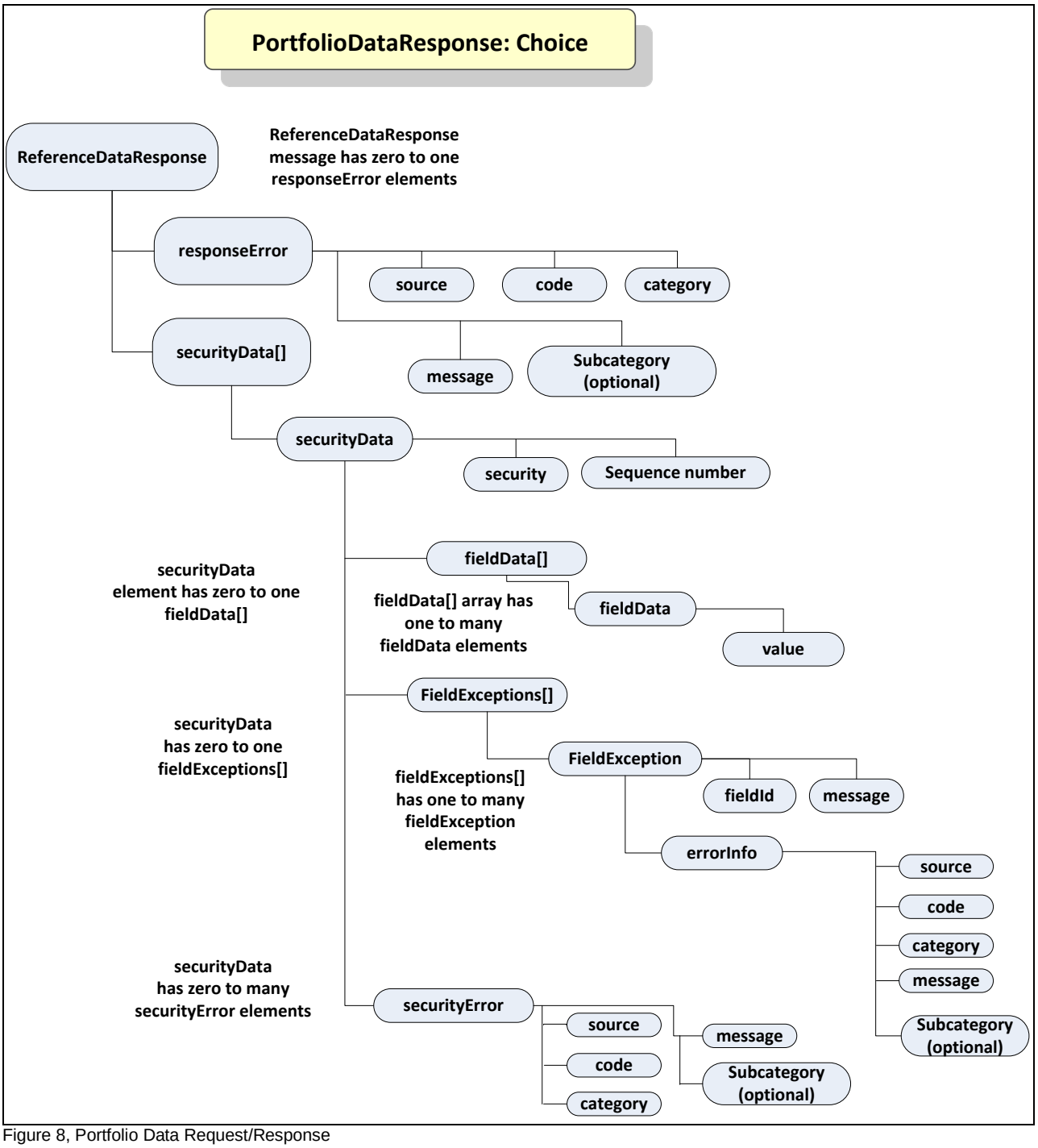


Figure 8, Portfolio Data Request/Response

6.11 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: Private or Global EQS 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");`			
Overrides: EQS information can also be accessed historically by using the PitDate override field and supplying the date in "yyyymmdd" format.			
Element	**Element Value**	**Type**	**Description**
fieldId		string	Field mnemonic "PiTDate"
value		string	Date in "yyyymmdd" format

Example Syntax:

```
Element overrides = request.getElement("overrides"); Element
override1 = overrides.appendElement();
override1.setElement("fieldId", "PiTDate");
override1.setElement("value", "20121210");
```

6.12 BEQSRESPONSE: CHOICE

The figure below shows the structure of a BEQSResponse. See "Reference Data Service Response" for more information.

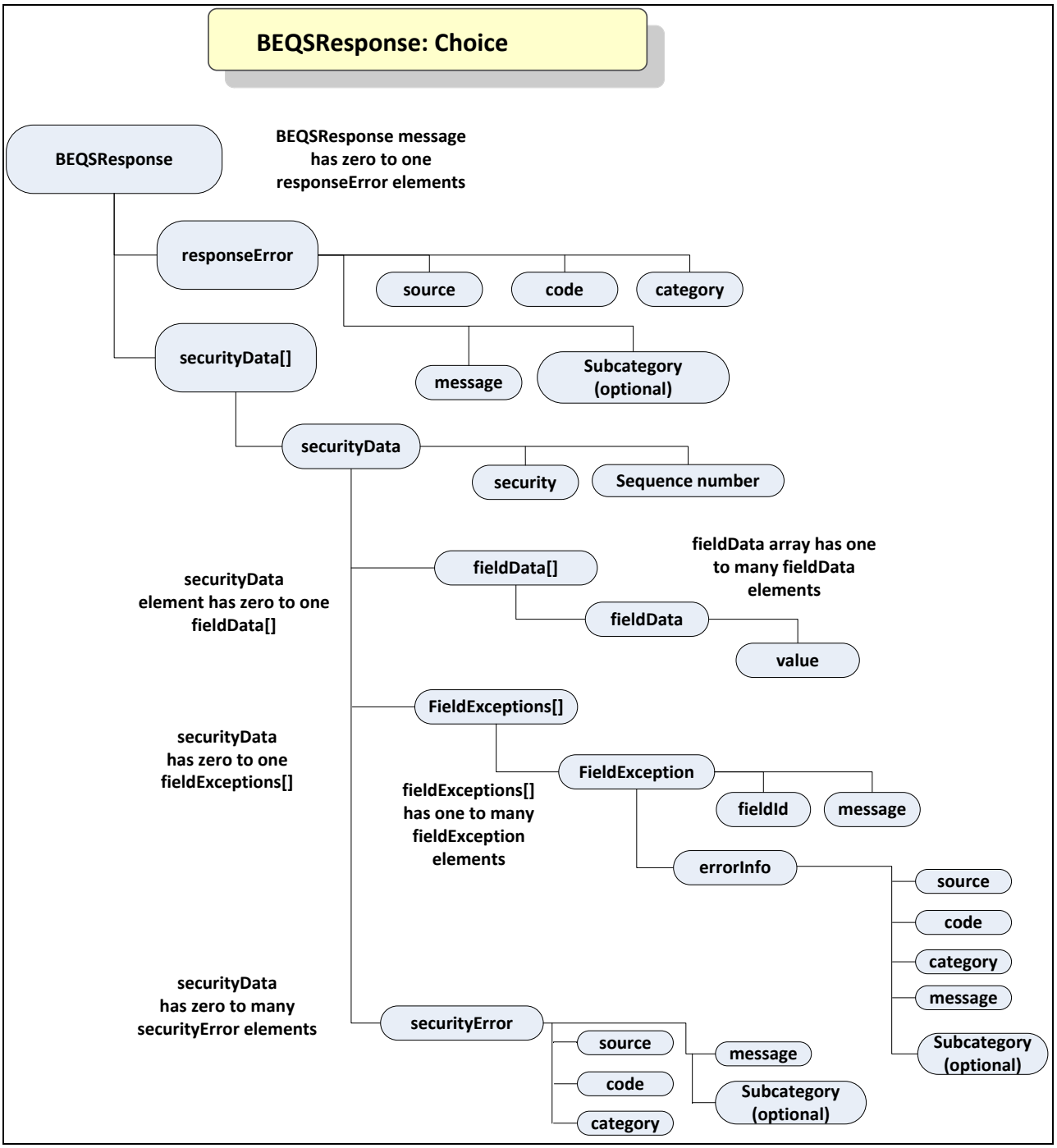


Figure 9. BEQS Response

6.13 REFERENCE DATA SERVICE RESPONSE

The two tables below give descriptions of the individual Elements received in a reference data response.

TABLE: 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 that occurred, including the source, code, category, Message and subcategory.

TABLE: REFERENCE DATA SERVICE RESPONSE VALUES

ELEMENT	TYPE	DESCRIPTION
security	string	The security requested.
eidData	integer	Entitlement identifier (EID) associated with requested security.
sequenceNumber	integer	Security sequence number; specifies the position of the security in the Request.
fieldId	string	Requested field represented as an alphanumeric or 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	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 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.
rpsCode	string	Transaction code. The following values appear: -B: A customer transaction where the dealer purchases securities from customer. -S: A customer transaction where the dealer sells securities to customer. -D: An inter-dealer transaction (always from the sell side).

brokerBuyCode	string	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 a broker's name, enter: MMID {market maker code} <GO>.
brokerSellCode	string	
micCode	string	The BIC, or bank identifier code is 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; it indicates the venue wherein the trade was executed.

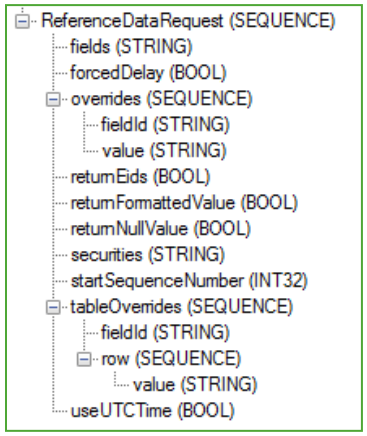
6.14 REFERENCE DATA VS. MARKET DATA

Reference Data

Nested structure vs. flat structure

Reference data in XML-like nested structure

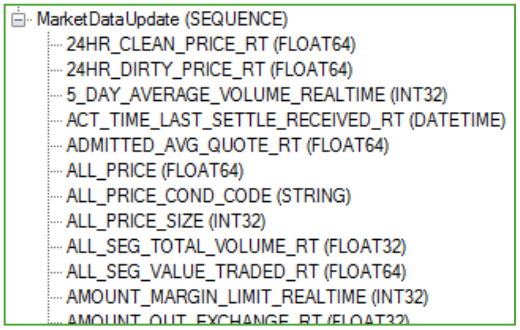
- Requests
- Response
  - Services: /refdata, /apiauth, /apiflds, etc.



Market Data

Market data results are flat, determined by Event type

- Subscriptions
- Events
  - Services: /mktdata, /mktvwap, /mktbar, etc.



☞ ***Note:*** *Market data Requests often return more fields than requested. All requested fields will be returned (if valid), but other fields are determined by the backend for performance reasons. Users should only rely on requested fields being returned.*

6.15 REQUESTING REFERENCE DATA

The ReferenceDataRequest Request type retrieves 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 by using the API fields service (covered later in this module).

A ReferenceDataRequest Request 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:

```
<C++>

// Assume the //blp/refdata service is already opened
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");
session.sendRequest(request, null);
```

Bulk fields and/or overrides can also be included in the Request. Because of the array-like format of a bulk field, they are processed a little differently (covered later in the guide).

6.16 HANDLING REFERENCE DATA MESSAGES

A RESPONSE Message will always be returned. For large requests, one or more PARTIAL_RESPONSE Event Messages will also be returned, which will include a subset of the information. A RESPONSE Message indicates the Request has been fully served. The example below shows how to process a Reference Data Response:

```
<C++>
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;
                        }
                    }
                }
            }
        }
    }
    private void processReferenceDataResponse(Message msg) throws
    Exception {
        MessageIterator msgIter(event);
        while (msgIter.next()) {
            Message msg = msgIter.message();
            Element securities = msg.getElement(SECURITY_DATA);
            size_t numSecurities = securities.numValues();
            std::cout << "Processing " << (unsigned int)numSecurities
    << " securities:"<< std::endl;
            for (size_t i = 0; i < numSecurities; ++i) {
                Element security = securities.getValueAsElement(i);
                std::string ticker =
    security.getElementAsString(SECURITY);
                std::cout << "\nTicker: " + ticker << std::endl;
                if (security.hasElement("securityError")) {
                    printErrorInfo("\tSECURITY FAILED: ",
    security.getElement(SECURITY_ERROR));
                    continue;
                }
                if (security.hasElement(FIELD_DATA)) {
                    const Element fields =
    security.getElement(FIELD_DATA);
                    if (fields.numElements() > 0) {
                        std::cout << "FIELD\t\tVALUE"<<std::endl;
                        std::cout << "-----\t\t-----"<< std::endl;
```

```
                        size_t numElements = fields.numElements();
                        for (size_t j = 0; j < numElements; ++j) {
                            Element field = fields.getElement(j);
                            std::cout << field.name() << "\t\t" <<
                            field.getValueAsString() << std::endl;
                        }
                    }
                }
                std::cout << std::endl;
            }
        }
    }
```

6.17 HANDLING REFERENCE DATA BULK MESSAGES

As discussed earlier, certain reference data fields are classified as bulk fields. These are indicated on “FLDS <GO>” with a “Show Bulk Data” Message, where the value would normally be displayed in the right-most column. An example bulk field would be “COMPANY_ADDRESS”. This field, as is the case with all of the API bulk fields, possesses more than one piece of information (e.g., the company’s full address).

To read a bulk response, additional processing must be implemented in the Event handler. The method below would be called once the data response was determined to contain bulk data; this is determined by checking to see if the field Element being returned is an array. Another way is to check is to see if the DataType of that field is a SEQUENCE type. Below is what the code might look like when determining if bulk data has been received:

```
<C++>
if (security.hasElement(FIELD_DATA)) {
    const Element fields = security.getElement(FIELD_DATA);
    if (fields.numElements() > 0) {
        cout << "FIELD\t\tVALUE"<<endl;
        cout << "-----\t\t-----"<< endl;
        size_t numElements = fields.numElements();
        for (size_t j = 0; j < numElements; ++j) {
            const Element  field = fields.getElement(j);
            // Checking if the field is Bulk field
            if (field.isArray()){
                processBulkField(field);
            }else{
                processRefField(field);
            }
        }
```

```
                }
        }
```

Below is the code for processBulkField needed to read the data from the bulk response:

```
<C++>

void processBulkField(Element refBulkfield)
{
    cout << endl << refBulkfield.name() << endl ;
    // Get the total number of Bulk data points
    size_t numofBulkValues = refBulkfield.numValues();
    for (size_t bvCtr = 0; bvCtr < numofBulkValues; bvCtr++) {
        const Element  bulkElement =
refBulkfield.getValueAsElement(bvCtr);
        // Get the number of sub fields for each bulk data element
        size_t numofBulkElements = bulkElement.numElements();
        // Read each field in Bulk data
        for (size_t beCtr = 0; beCtr < numofBulkElements; beCtr++){
            const Element  elem = bulkElement.getElement(beCtr);
            cout << elem.name() << "\t\t"
<< elem.getValueAsString() << endl;
        }
    }
}
```

6.18 HANDLING HISTORICAL DATA MESSAGES

A successful HistoricalDataResponse (with no errors or exceptions) holds information on a single security. It contains a HistoricalDataTable with one HistoricalDataRow for each interval returned.

```
<C++>
while (true)
{
    Event event = session.nextEvent();
    MessageIterator msgIter(event);
    while (msgIter.next())
    {
        Message &msg = msgIter.message();
        if ((event.eventType() != Event::PARTIAL_RESPONSE) &&
            (event.eventType() != Event::RESPONSE))
        {
            continue;
        }
```

```
            Element securityData = msg.getElement(SECURITY_DATA);
            Element securityName =
    securityData.getElement(SECURITY_NAME);
            std::cout << securityName << "\n\n";

             //only process field data if no errors or exceptions have
    occurred
            if(!ProcessExceptions(msg))
            {
                if(!ProcessErrors(msg))
                {
                    ProcessFields(msg);
                }
            }
            std::cout << "\n\n";
        }
        if (event.eventType() == Event::RESPONSE) {
                    break;
        }
    }
```

In the above while() loop, if it has no exceptions or errors, the ProcessFields function is called. To see the code for this function, look at the HistoryExample C++ example, which is found in the Server C++ API SDK installation. Currently, this example is not available in the B-PIPE SDK.

6.19 COMBINING REFERENCE AND SUBSCRIPTION DATA

When developing an application that will handle real-time streaming and static data, a separate Session can be used for each type of data. This is to ensure that the processing of a heavyweight

Subscription, for instance, is not being slowed by the reading and blocking of multiple static Request responses. In fact, a Subscription might have a separate Session, with another for frequent reference data Requests (for fields unavailable in a real-time format) and still another for occasional large intraday type Requests.

7 Volume-Weighted Average Price (//blp/vwap)

The custom volume-weighted average price (VWAP) service ("`//blp/mktvwap`") provides streaming VWAP values for equities. This service allows for a customized data stream with a series of overrides, as outlined in the API "Developer's Guide".

Following is a sample custom market VWAP string:

```
//blp/mktvwap/ticker/IBM US Equity?fields=VWAP&VWAP_START_TIME=10:00&VWAP_END_TIME=16:00
```

Notice that it includes a single main field ("VWAP") and two override field/value pairings (VWAP_START_TIME=10:00 and VWAP_END_TIME=16:00).

User can select the single topic overload of the ADD method and pass the entire string formulated above or break down the string into topic, fields and overrides—and use that applicable overload of the ADD method.

The following code sample demonstrates how this can be accomplished. The Response will return a Message containing a selection of VWAP fields.

```
<C++>
// Assume that session already exists and "//blp/mktvwap" service
// has been opened.
SubscriptionList subscriptions;
subscriptions.add("//blp/mktvwap/ticker/IBM US Equity",
                  "VWAP",
                  "VWAP_START_TIME=10:00&VWAP_END_TIME=16:00"
                  CorrelationId(10));
session.subscribe(subscriptions);
```

7.1 VWAP SCHEMA — 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) );`		

USEUTC	Boolean	Setting to true returns values in UTC. Setting to false causes default to the TZDF <GO> settings of the requestor.

8 API Field Service (//blp/apiflds)

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 List Request**: Provides a full list of fields as specified by the field type (e.g., All, Static or RealTime).
- **Field Information Request**: Provides a description of the specified fields in the request.
- **Field Search Request**: Provides the ability to search the Bloomberg data model with a search string for field mnemonics.
- **Categorized Field Search Request**: Provides the ability to search the Bloomberg data model based on categories with a search string for field mnemonics.

Listed below is the schema for API field service `//blp//apiflds`:

8.1 REQUESTS: CHOICE

It is the top-level Request to the service.

Element	Type	Description
fieldInfoRequest	FieldInfoRequest	Request for field information
fieldSearchRequest	FieldSearchRequest	Field search information
categorizedFieldSearchRequest	CategorizedFieldSearchRequest	

8.2 RESPONSES: CHOICE

This is the top-level Request to the service.

Element	Type	Description
fieldResponse	FieldResponse	Field response information
categorizedFieldResponse	CategorizedFieldResponse	

8.3 FIELD INFORMATION REQUEST

Identifier: Reference or streaming fields desired.			
Element	**Element Value**	**Type**	**Description**
id		string	Fields can be specified as an 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
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);`			

8.3.1 FIELD INFORMATION REQUEST RESPONSE

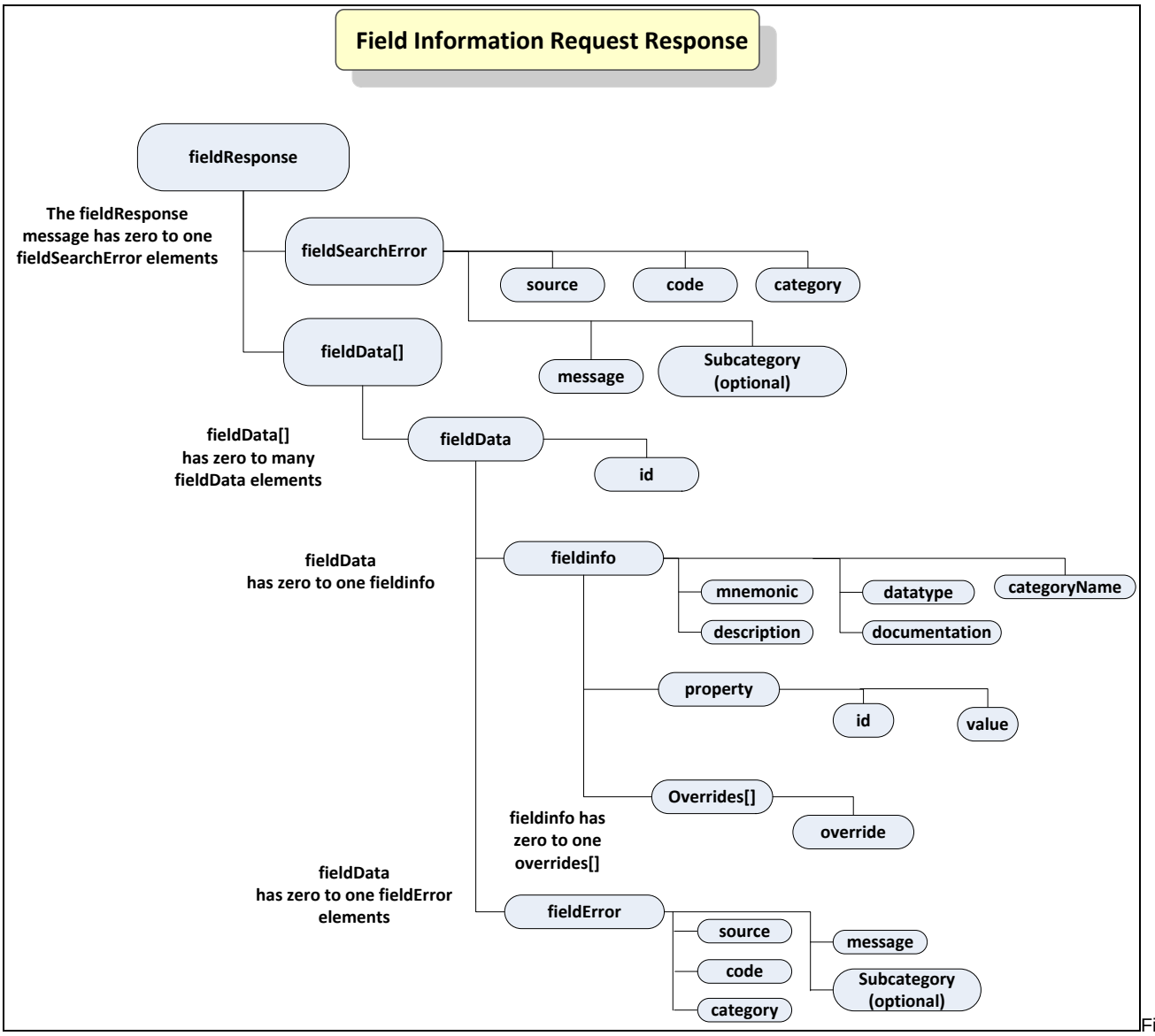


Figure 10. Field Information Request Response

8.3.2 FIELD SEARCH REQUEST

Identifier: 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	Results filtered by fields available 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 both streaming fields (real-time and delayed) and reference fields (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	Results filtered by fields available 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 both streaming fields (real-time and delayed) and reference fields (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);`

8.3.3 FIELD SEARCH REQUEST RESPONSE

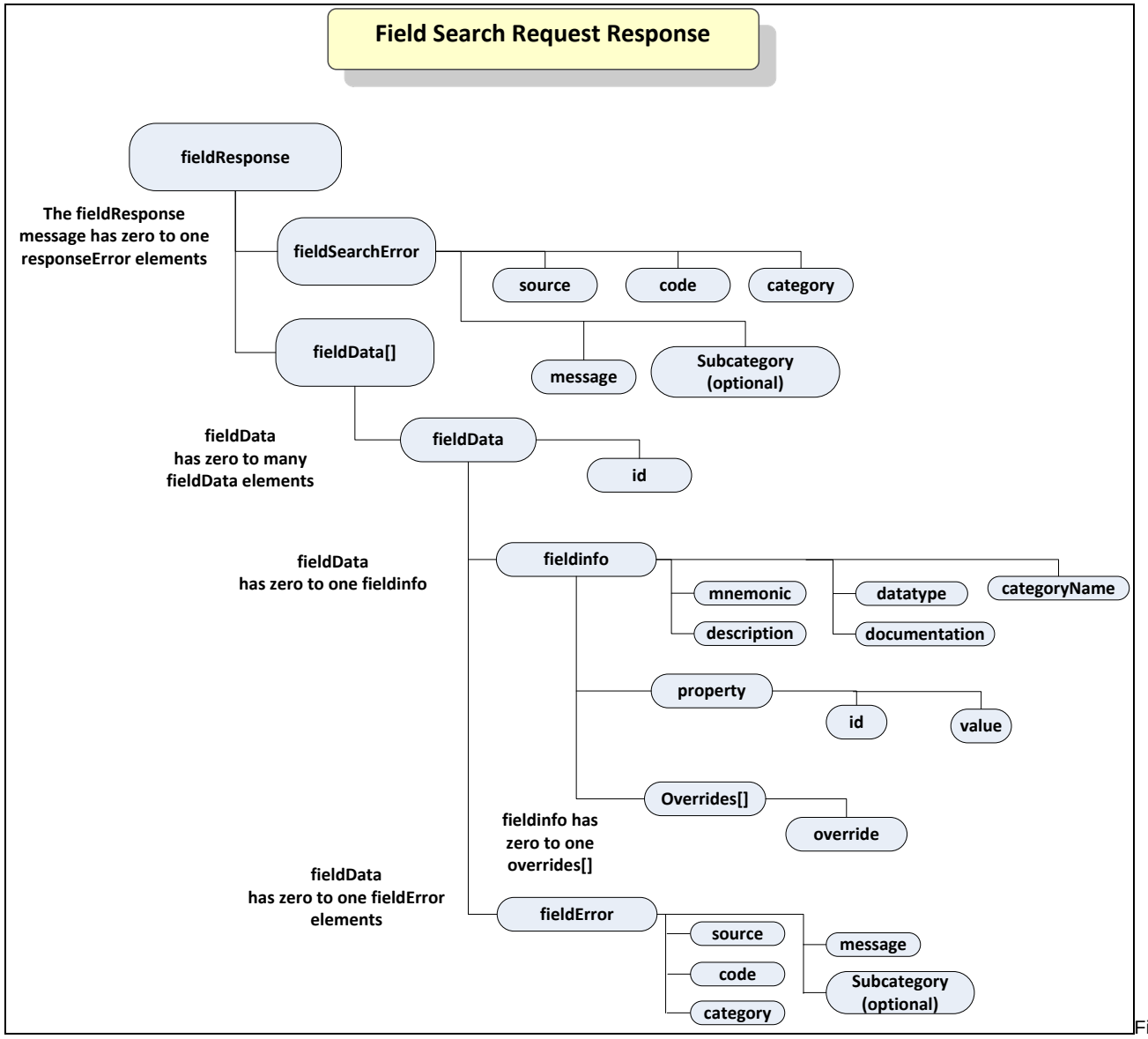


Figure 11. Field Search Request Response

8.3.4 CATEGORIZED FIELD SEARCH REQUEST

Identifier: 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	Results filtered by fields available 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	sstring	Results include both streaming fields (real-time and delayed) and reference fields (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 description of the field as seen on FLDS <GO>. Default value is false.

Example Syntax:

```
request.Set("returnFieldDocumentation", true);
```

8.3.5 CATEGORIZED FIELD SEARCH REQUEST RESPONSE

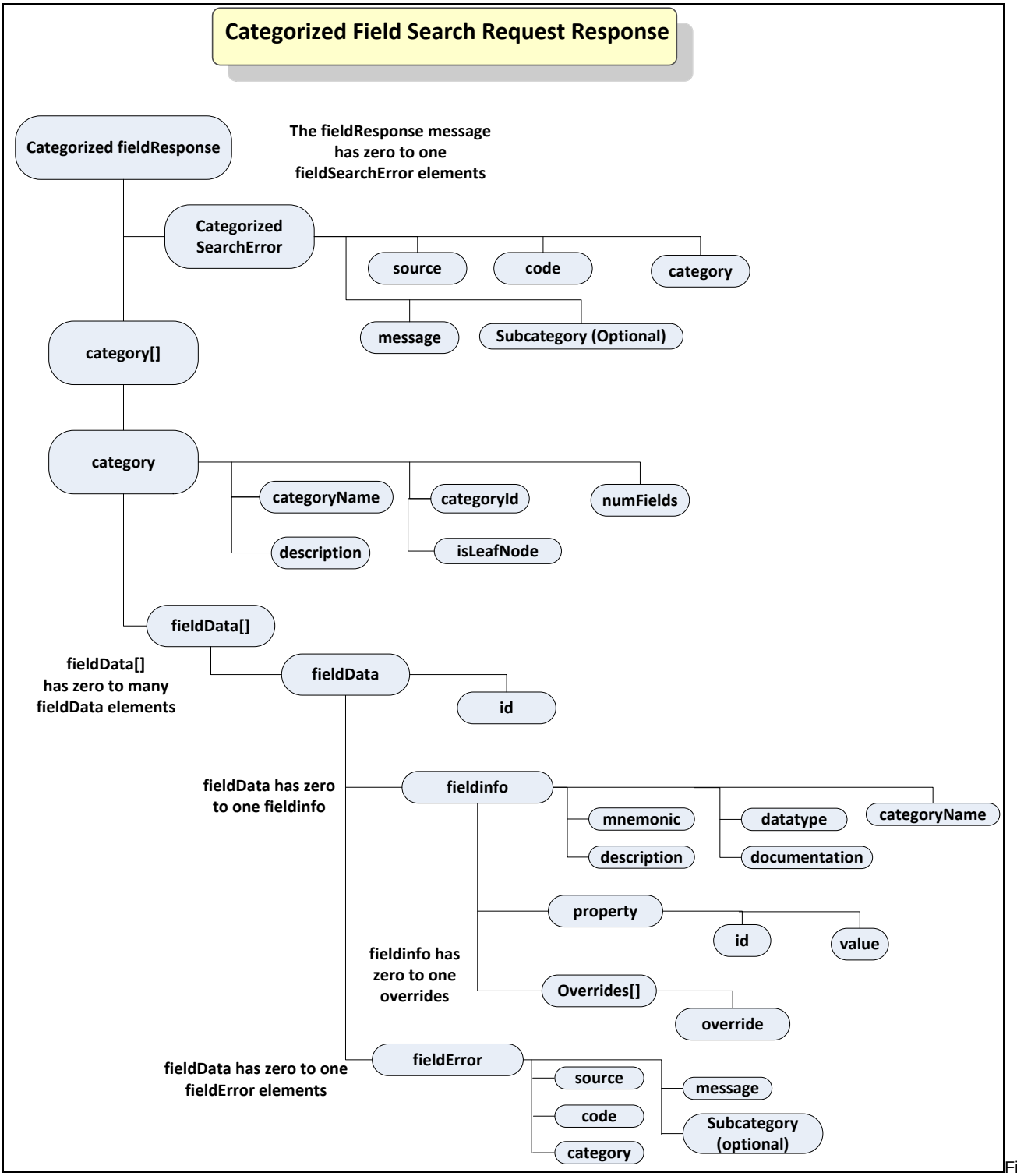


Figure 12. Categorized Field Search Request Response

8.3.6 FIELD LIST REQUEST

Identifier: Reference or streaming fields desired			
Element	**Element Value**	**Type**	**Description**
fieldType	All	String	Results include both streaming (real-time and delayed) and reference (static) fields.
	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**
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);`			

8.3.6.1 A.3.6.1 FIELD LIST REQUEST RESPONSE

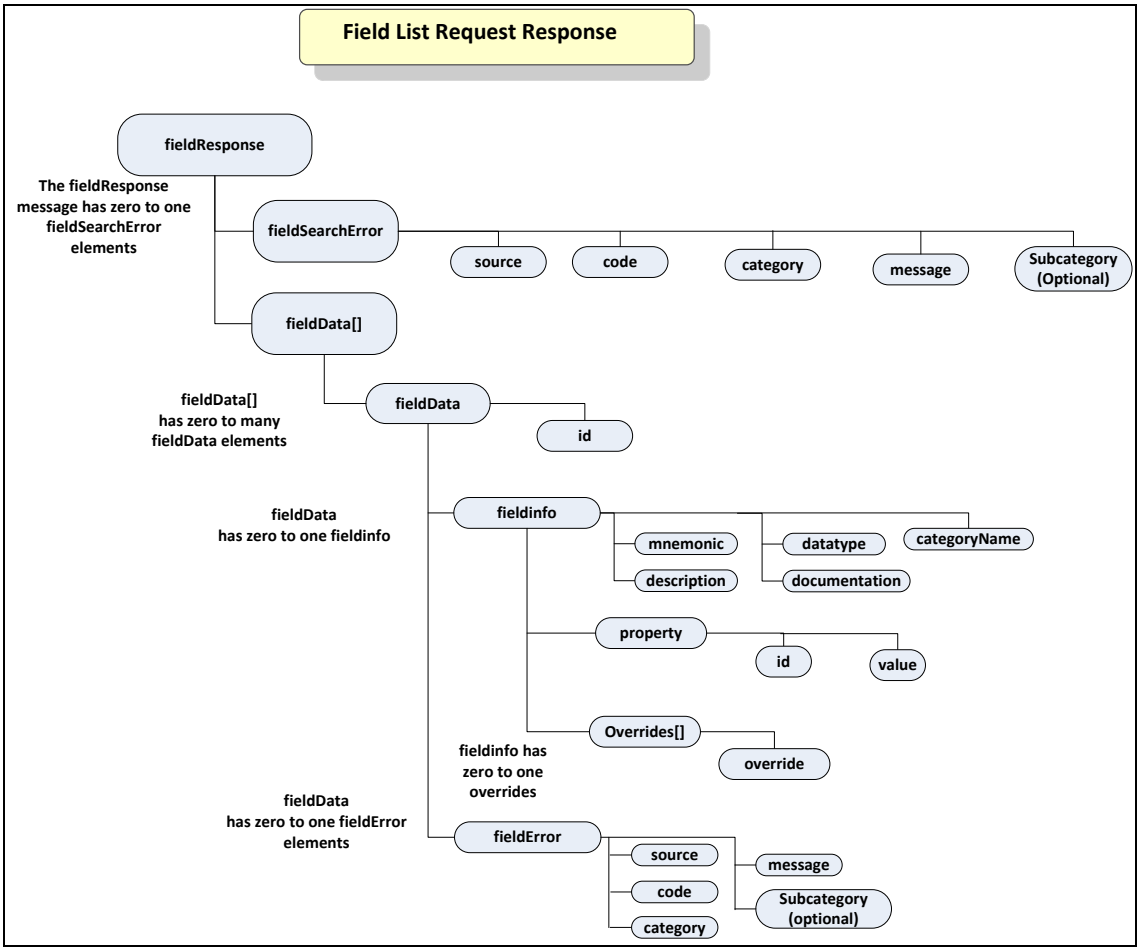


Figure 13. Field List Request Response

8.3.7 FIELD SERVICE RESPONSE ELEMENTS

The following table provides descriptions of the individual Elements received in the field service responses. Please see 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 that has occurred, including the source, code, category, Message and subcategory.

8.3.8 FIELD SERVICE RESPONSE VALUES

Element	Type	Description
id	string	Resulting field represented as an alphanumeric or a mnemonic, e.g., PR005 or PX_LAST.
mnemonic	integer	Resulting field represented as a mnemonic, e.g., 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, Trading Systems.
description	string	IThe short description of 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	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.

8.4 API FIELD SERVICE — FIELD LIST

A FieldListRequest Request returns all of the fields defined by the field type. This loosely follows the field type filter option available on “FLDS <GO>” on the Bloomberg Professional service.

The example below shows how to construct a FieldListRequest Request.

```
<C++>
Service fieldInfoService =
session.getService("//blp/apiflds");
Request request =
fieldInfoService.createRequest("FieldListRequest");
request.append("fieldType", "All");  // Other options are
Static and RealTime
request.set("returnFieldDocumentation", true);
std::cout << "Sending Request: " << request << std::endl;
session.sendRequest(request);
```

Possible fieldType values include “All” (to return all fields in the API Data Dictionary), “Static” (to return all static fields contained in the API Data Dictionary) and “RealTime” (to return all real-time fields contained in the API Data Dictionary).

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. It is assumed that an Event was received with either a RESPONSE or PARTIAL_RESPONSE type prior to running this processFieldResponse method:

```
<C++>
MessageIterator msgIter(event);
while (msgIter.next()) {
    Message msg = msgIter.message();
    Element fields = msg.getElement("fieldData");
    int numElements = fields.numValues();
    printHeader();
    for (int i=0; i < numElements; i++) {
        printField (fields.getValueAsElement(i));
    }
    std::cout << std::endl;
}
if (event.eventType() == Event::RESPONSE) {
    break;
}
```

8.5 API FIELD SERVICE — FIELD INFORMATION

A FieldInfoRequest Request 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. The Request can also specify the return of the documentation as per the “FLDS <GO>” function.

This example shows how to construct a FieldInfoRequest Request.

```
<C++>
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);
std::cout << "Sending Request: " << request << std::endl;
session.sendRequest(request);
```

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. It is assumed that an Event was received with either a RESPONSE or PARTIAL_RESPONSE type prior to running this processFieldResponse method:

```
<C++>
MessageIterator msgIter(event);
while (msgIter.next()) {
    Message msg = msgIter.message();
    Element fields = msg.getElement("fieldData");
    int numElements = fields.numValues();
    printHeader();
    for (int i=0; i < numElements; i++) {
        printField (fields.getValueAsElement(i));
    }
    std::cout << std::endl;
}
if (event.eventType() == Event::RESPONSE) {
    break;
}
```

The above code snippet does not provide the code for either the printField or printHeadermethods. To view these, refer to the SimpleFieldInfoExample example installed with the C++ API SDK.

8.6 API FIELD SERVICE — FIELD SEARCH

A FieldSearchRequest Request returns a list of fields matching a specified search criterion. The Request specifies a search string and may also contain criteria to filter the results. Thesw criteria allow for the filtering by category, product type and field type. Detailed information on these settings is found in “Appendix A: Schemas” in the “API Developer’s Guide”.

The following example shows how to construct a FieldSearchRequest Request:

```
<C++>
Service fieldInfoService =
session.getService("//blp/apiflds");
Request request =
fieldInfoService.createRequest("FieldSearchRequest");
request.append("searchSpec", "last price");
Element exclude = request.getElement("exclude")
Exclude.setElement("fieldType", "Static");
std::cout << "Sending Request: " << request << std::endl;
session.sendRequest(request);
A FieldSearchRequest returns a FieldResponse just as a
FieldInfoRequest does. It is assumed that an Event was
received with either a RESPONSE or PARTIAL_RESPONSE type
prior to running this processFieldResponse method:
<C++>
MessageIterator msgIter(event);
while (msgIter.next()) {
     Message msg = msgIter.message();
     Element fields = msg.getElement("fieldData");
    int numElements = fields.numValues();
    for (int i=0; i < numElements; i++) {
        printField (fields.getValueAsElement(i));
    }
    std::cout << std::endl;
}
if (event.eventType() == Event::RESPONSE) {
    break;
}
```

The above code snippet does not provide the code for the printFieldmethod. To view this, refer to the SimpleFieldSearchExample example installed with the C++ API SDK.

8.7 API FIELD SERVICE — CATEGORIZED FIELD SEARCH

A CategorizedFieldSearchRequest Request returns a list of fields matching a specified set of search criteria. The Request specifies a search string and may also contain criteria to filter the results. These criteria allow for the filtering by category, product type and field type.

☞ *For additional information, refer to the "Core User Guide".*

The following example shows how to construct a CategorizedFieldSearchRequest Request:

```
<C++>
Service fieldInfoService = session.getService("//blp/apiflds");
Request request =
fieldInfoService.createRequest("CategorizedFieldSearchRequest");
request.append("searchSpec", "last price");
Element exclude = request.getElement("exclude")
Exclude.setElement("fieldType", "Static");
Request.set("returnFieldDocumentation", false);
std::cout << "Sending Request: " << request << std::endl;
session.sendRequest(request);
```

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.

```
<C++>
MessageIterator msgIter(event);
while (msgIter.next()) {
    Message msg = msgIter.message();
    if (msg.hasElement(FIELD_SEARCH_ERROR)) {
        msg.print(std::cout);
        continue;
    }
    Element categories = msg.getElement("category");
    int numCategories = categories.numValues();
    for (int catIdx=0; catIdx < numCategories; ++catIdx) {
       Element category =
categories.getValueAsElement(catIdx);
       std::string Name =
category.getElementAsString("categoryName");
       std::string Id =
category.getElementAsString("categoryId");
```

```
            std::cout << "\n  Category Name:" << padString
    (Name, CAT_NAME_LEN) <<
                          "\tId:" << Id << std::endl;
            Element fields = category.getElement("fieldData");
            int numElements = fields.numValues();
            printHeader();
            for (int i=0; i < numElements; i++) {
                 printField (fields.getValueAsElement(i));
            }
        }
        std::cout << std::endl;
    }
    if (event.eventType() == Event::RESPONSE) {
        break;
    }
```

9 Security Lookup (//blp/instruments)

The Instruments service is used to perform three types of operations: 1. a Security Lookup Request; 2. a Curve Lookup Request; 3. a Government Lookup Request. 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 have EID 14005 (if requested by someone with level 1 access).

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

9.1 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 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, users get results translated into the specified language.

The following code snippet demonstrates how to make a Security Lookup Request, 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);
```

9.2 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, 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);
```

9.3 GOVERNMENT LOOKUP REQUEST

The Government Lookup Request searches through government securities. As with all Requests, users can specify the "query" string and the maximum number of results. As every government security has a Ticker that is not unique, these securities can also be filtered by this Ticker. For example, a user can specify filter Tickers equal to "T" or set partial match (i.e., "partialMatch") to true and filter out all government securities beginning with the letter "T" by setting the "query" element value to "T*".

The following code snippet demonstrates how to make a government lookup Request, 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);
```

9.4 RESPONSE BEHAVIORS

Each lookup response will comprise zero or more PARTIAL_RESPONSE Event types and one RESPONSE Event type event — which users will be familiar with if they 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;
    }
    }
    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;
    }
    }
```

```
    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;
    }
    }
```

10 Real-time and Delayed Intraday Bars (//blp/mktbar)

10.1 MARKET BAR SUBSCRIPTION SERVICE

The market bar service is Subscription-based service that provides streaming (real-time and delayed) intraday bars. This service allows for bucketized data stream where each bucket (“bar”) consists of the following aspect fields:

time	low	value
open	close	volume
high	number of ticks	datetime

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

TOPIC STRING:
“//blp/mktbar/*SYMBOLOGY*/*SECURITY*?START_TIME=*start*&END_TIME=*end*&BAR_SIZE=*size*”

FOR EXAMPLE:
“//blp/mktbar/ticker/VOD LN Equity?start_time=9:30&bar_size=10”

The MKTBAR service is based on TRADE ticks only. Hence, the Subscription topic string should have the option “fields=LAST_PRICE”. The following code snippet shows a Subscription to market bars:

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

```
    new CorrelationID("IBM US Equity")));
    d_session.subscribe(d_subscriptions);
```

RESPONSE BEHAVIOR

Successful Subscription to MKTBAR service will result in the following types of Messages being sent to subscriber:

- MarketBarStart
- MarketBarUpdate
- MarketBarIntervalEnd
- MarketBarEnd

MarketBarStart is generated upon every new bar; therefore, the frequency of this Event depends on the bar_size setting and the fact that security is active at the time. A MarketBarStart Event returns all fields of the bar with values filled in since the start of the bar until Subscription time. Subsequently, on every TRADE update, a MarketBarUpdate is sent.

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

MarketBarIntervalEnd is sent at the end of each bar and always precedes next MarketBarStart. This Message contains only TIME and DATE.

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

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

Please note that no initial summary is returned for streaming intraday bars for a start date earlier than now. A Reference Data intraday bar Request is required before a Subscription to get an initial snapshot if needed.

When a market bar Subscription is set to return delayed data, the market bar start Message is not returned until the delayed period has passed.

10.2 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`"
Fields	string	MKTBAR service is based on TRADE ticks only. Hence, Subscription topic string should have option "`fields=LAST_PRICE`". Fields can be specified as an alpha numeric or mnemonic.
bar_size	string	Length of bar defined in minutes. The minimum supported size of the bar is 1 min. The maximum supported size of the bar is 1,440 minutes, (=24 hours).
start_time	string	Optional. Should be in format hh:mm. If not, set the time of Session start of the security or Subscription time is used.
end_time	string	Optional. Should be in format hh:mm. If not specified, then security's Session end time is used.

Example Syntax:

```
Subscription mySubscription = new Subscription(
"//blp/mktbar/TICKERX/IBM US Equity",
"last_price",
"bar_size=5&start_time=13:30&end_time=20:00",
new CorrelationID("IBM US Equity"));
```

10.3 MARKET BAR SUBSCRIPTION: DATA EVENTS RESPONSE

Each bar update includes two time fields: TIME and DATE_TIME. Both of datetime type. While TIME carries the time of the current bar, DATE_TIME also includes the date of the bar— thereby indicating the date change if Subscription left running overnight.

MARKETBARSTART

```
/blp/mktbar/TICKER/IBM US Equity - MarketBarStart
```

TIME = 12:5

OPEN = 176.88

HIGH = 176.89

LOW = 176.85

CLOSE = 176.88

NUMBER_OF_TICKS = 12

VOLUME = 1400

VALUE = 247622.0 DATE_TIME = 2/7/2014 12:5

MARKETBARUPDATE

`//blp/mktbar/TICKER/IBM US Equity - MarketBarUpdate`

TIME = 12:5

HIGH = 176.89

LOW = 176.85

CLOSE = 176.87

NUMBER_OF_TICKS = 13

VOLUME = 1500

VALUE = 265309.0 DATE_TIME = 2/7/2014 12:5

MARKETBARINTERVALEND

`//blp/mktbar/TICKER/IBM US Equity - MarketBarIntervalEnd`

TIME = 12:5

DATE_TIME = 2/7/2014 12:5

MARKETBAREND

`//blp/mktbar/TICKER/IBM US Equity - MarketBarEnd`

TIME = 12:5

DATE_TIME = 2/7/2014 12:5

Argument Value	Type	Description
TIME	datetime	Returns time of start of bar bucket.
Example Syntax: `Datetime time = msg.getElementAsDatetime(TIME);`		
OPEN	Float64	Returns open price of bar bucket. Should be returned in the `MarketBarStart` Event.
Example Syntax: `int open = msg.getElementAsFloat64(OPEN);`		
HIGH	Float64	Returns high price of bar bucket in `MktBarStart` and subsequently in every `MktBarUpdate` if higher price occurs until the end of the bar.
Example Syntax: `int high = msg.getElementAsFloat64(HIGH);`		
LOW	Float64	Returns low price of bar bucket in `MktBarStart` and subsequently in every `MktBarUpdate` if lower price occurs until the end of the bar.
Example Syntax: `int low = msg.getElementAsFloat64(LOW);`		
CLOSE	Float64	Returns every updated close price between `MktBarStart` and `MktBarUpdate` Event.
Example Syntax: `int close = msg.getElementAsFloat64(CLOSE);`		
NUMBER_OF_TICKS	Int32	Accumulates number of ticks in bar on every `MktBarStart` and `MktBarUpdate` Event till `MarketBarIntervalEnd` sent.

Example Syntax: `int number_of_ticks = msg.getElementAsInt32(NUMBER_OF_TICKS);`		
VALUE	Float64	Volume*Price increments for number of trades in each market bar; is reset at the start of each market bar.
Example Syntax: `float value = msg.getElementAsInt64(VALUE);`		
VOLUME	Int64	Volume increments for number of trades in each market bar; is reset at the start of each market bar.
Example Syntax: `float volume = msg.getElementAsInt64(VOLUME);`		
DATE_TIME	datetime	Returns date and time of bar bucket. NOTE: value of the field consists of MM/DD/YYY HH:MM.
Example Syntax: `Datetime datetime = msg.getElementAsDatetime(DATE_TIME);`		

11 B-PIPE-Only Services

This section will expand on each of the four services specific to B-PIPE developers only.

B-PIPE provides access to the full list of current bid and ask prices for an instrument; this list can be known as market depth, order books or simply “level 2” data. Most Exchanges will consider this to be a separate product from their “level 1” data (general real-time) and will charge additional fees for access to it. Thus a different EID is typically used for “level 2”. The services are as follows:

- Market Depth Service (`//blp/mktdepthdata`)
- Market List Service (`//blp/mktlist`)
- Source Reference Service (`//blp/srcref`)
- Message Scraping Service (`//blp/msgscrape`)

Field filtering is available as a configuration option—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.

11.1 DEPTH OF BOOK SERVICE (//BLP/MKTDEPTHDATA)

The Enterprise Market Depth Service (EMDS) is Subscription-based and allows users to access a more comprehensive set of market-depth data for supported and entitled securities. It is available to both BPS (Bloomberg Professional service) and non-BPS users.

B-PIPE provides access to the full list of current bid and ask prices for an instrument; this list can be known as market depth, order books, or simply “level 2” data. Most exchanges will consider this to be a separate product from their “level 1” data (general real-time) and will charge additional fees for access to it. Thus a different EID is typically used for “level 2”. Market depth, order books and level 2 data are all names for the same set of data.

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. The best bid in the order book should generally be lower than the best ask, but it is possible for the ask to be higher than the bid. In that case, it is known as a “crossed” or “inverted” market (or book). The details of the specific conditions vary by market.

Many times exchanges consider order book (level 2) information to be 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 defining characteristics: 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 three types of order books, Market-By-Order (MBO); Market-By-Level (MBL); and Market Maker Quote (MMQ). An Exchange that operates an order book may provide only MBL data, only MBO data or both MBO and MBL data. An Exchange that operates a market-maker quote book will provide MMQ data. The three

order/quote book update methods: Replace-By-Position (RBP); Add-Mod-Delete (AMD); and Replace-By-Broker (RBB).

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 `//blp/mktdata` service. This service provides up to the first 10 levels of market depth by level (aka MBL) data. To get this data, make a `//blp/mktdata` Subscription and include one or more of the following fields:

Mnemonic	Description
BEST_BID1 thru BEST_BID10	First thru tenth best bid price in 10 levels of market depth
BEST_BID1_SZ thru BEST_BID10_SZ	Size of first thru tenth best bid in 10 levels of market depth
BEST_ASK1 thru BEST_ASK10	First thru tenth best ask price in 10 levels of market depth
BEST_ASK1_SZ thru BEST_ASK10_SZ	Size of first thru tenth best ask in 10 levels of market depth

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 users to obtain market by order (MBO) data. Also, the `//blp/mktdata` service does not provide information such as the book type or the action performed on that position.

Therefore, if users wish to receive more than 10 levels of MBL or any MBO levels, they will be required to use the `//blp/mktdepthdata` service. Subscribing to this comprehensive service will both supply them with the order book in its entirety and also provide the book type, action performed, etc.

11.1.1 CODE EXAMPLES

11.1.1.1 MARKETDEPTHSUBSCRIPTION EXAMPLE

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/mktdepthdata" service has been successfully opened.

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

The following code snippet details how to handle and print out a MarketDepth Subscription to std::cout. This C++ snippet is based on the above “MarketDepthSubscriptionExample” C++ SDK example. For a more complete example that demonstrates how to handle and build an order/level book, please reference the “MarketDepthSubscriptionSnaphotExample” example in either the Java, C++ or .NET SDK.

```
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;
}
```

11.1.1.2 MARKETDEPTHSUBSCRIPTIONSNAPSHOT EXAMPLE

This example shows how to build and update an order and level book. It is of the LevelBook and OrderBook class, which handle the market-depth-by-level and by-order Messages, respectively.

C++

```
// ByOrderBook             d orderBooks[size];
// ByLevelBook             d_levelBooks[size];
// SubscriptionList        d subscriptions;
namespace
{
  const Name MKTDEPTH_EVENT_SUBTYPE("MKTDEPTH_EVENT_SUBTYPE");
  const Name MD GAP DETECTED("MD GAP DETECTED");
  const Name MD TABLE CMD RT("MD TABLE CMD RT");
  const Name MD_BOOK_TYPE("MD_BOOK_TYPE");
  const Name MD_MULTI_TICK_UPD_RT("MD_MULTI_TICK_UPD_RT");
  const Name MBO_WINDOW_SIZE("MBO_WINDOW_SIZE");
  const Name MBO_ASK_POSITION_RT("MBO_ASK_POSITION_RT");
  const Name MBO_ASK_RT("MBO_ASK_RT");
  const Name MBO ASK BROKER RT("MBO ASK BROKER RT");
  const Name MBO_ASK_COND_CODE_RT("MBO_ASK_COND_CODE_RT");
  const Name MBO_ASK_SIZE_RT("MBO_ASK_SIZE_RT");
  const Name MBO_TABLE_ASK("MBO_TABLE_ASK");
  const Name MBO BID POSITION RT("MBO BID POSITION RT");
  const Name MBO_BID_RT("MBO_BID_RT");
  const Name MBO BID BROKER RT("MBO BID BROKER RT");
  const Name MBO_BID_COND_CODE_RT("MBO_BID_COND_CODE_RT");
  const Name MBO_BID_SIZE_RT("MBO_BID_SIZE_RT");
  const Name MBO_TABLE_BID("MBO_TABLE_BID");
  const Name MBO_TIME_RT("MBO_TIME_RT");
  const Name MBO_SEQNUM_RT("MBO_SEQNUM_RT");
  const Name MBA WINDOW SIZE("MBA WINDOW SIZE");
  const Name MBA_ASK_POSITION_RT("MBA_ASK_POSITION_RT");
  const Name MBA_ASK_RT("MBA_ASK_RT");
  const Name MBA_ASK_NUM_ORDERS_RT("MBA_ASK_NUM_ORDERS_RT");
  const Name MBA_ASK_COND_CODE_RT("MBA_ASK_COND_CODE_RT");
  const Name MBA_ASK_SIZE_RT("MBA_ASK_SIZE_RT");
  const Name MBA TABLE ASK("MBA TABLE ASK");
  const Name MBA_BID_POSITION_RT("MBA_BID_POSITION_RT");
  const Name MBA_BID_RT("MBA_BID_RT");
  const Name MBA_BID_NUM_ORDERS_RT("MBA_BID_NUM_ORDERS_RT");
  const Name MBA_BID_COND_CODE_RT("MBA_BID_COND_CODE_RT");
  const Name MBA BID SIZE RT("MBA BID SIZE RT");
  const Name MBA_TABLE_BID("MBA_TABLE_BID");
  const Name MBA_TIME_RT("MBA_TIME_RT");
  const Name MBA_SEQNUM_RT("MBA_SEQNUM_RT");
  const Name NONE("NONE");

  const Name ADD("ADD");
  const Name DEL("DEL");
  const Name DELALL("DELALL");
  const Name DELBETTER("DELBETTER");
  const Name DELSIDE("DELSIDE");
  const Name EXEC("EXEC");
  const Name MOD("MOD");
  const Name REPLACE("REPLACE");
  const Name REPLACE BY BROKER("REPLACE BY BROKER");
  const Name CLEARALL("CLEARALL");
  const Name REPLACE_CLEAR("REPLACE_CLEAR");
  const Name REPLACE_BY_PRICE("REPLACE_BY_PRICE");

  const Name ASK("ASK");
  const Name BID("BID");
  const Name ASK_RETRANS("ASK_RETRANS");
  const Name BID_RETRANS("BID_RETRANS");
  const Name TABLE_INITPAINT("TABLE_INITPAINT");
```

```
    const Name TABLE_UPDATE("TABLE_UPDATE");

    const int BIDSIDE = 0;
    const int ASKSIDE = 1;

    const int BYORDER = 0;
    const int BYLEVEL = 1;

    Name PRICE_FIELD[2][2] = {MBO_BID_RT, MBO_ASK_RT, MBA_BID_RT, MBA_ASK_RT};
    Name    SIZE FIELD[2][2]    =    {MBO BID SIZE RT,    MBO ASK SIZE RT,    MBA BID SIZE RT,
MBA_ASK_SIZE_RT};
    Name POSITION_FIELD[2][2] = {MBO_BID_POSITION_RT, MBO_ASK_POSITION_RT, MBA_BID_POSITION_RT,
MBA_ASK_POSITION_RT};
    Name ORDER FIELD[2][2] = {NONE, NONE, MBA BID NUM ORDERS RT, MBA ASK NUM ORDERS RT} ;
    Name BROKER_FIELD[2][2] = {MBO_BID_BROKER_RT, MBO_ASK_BROKER_RT, NONE, NONE};
    Name TIME FIELD[2] = {MBO TIME RT, MBA TIME RT};
}

/*-------------------------------------------------------------------------------------
 * Name        : processSubscriptionDataEvent
 * Description : process market depth data events
 * Arguments   : event is the data event
 *             : session is the API session
 * Returns     : none
 *-------------------------------------------------------------------------------------*/
bool processSubscriptionDataEvent(const Event& event, Session* session)
{
    char timeBuffer[64];
    getTimeStamp(timeBuffer, sizeof(timeBuffer));

    MessageIterator msgIter(event);
    while (msgIter.next())
    {
      Message msg = msgIter.message();
      const char* msg_type = msg.messageType().string();
      if (strcmp(msg_type,"MarketDepthUpdates") == 0)
      {
        // Market Depth data
        if (d showTicks > 0)
        {
          // output tick message
          std::cout << timeBuffer << ": ";
          printFragType(msg.fragmentType());

          msg.print(std::cout);
          std::cout << std::flush;
        }

        // process base on book type
        switch (d marketDepthBook)
        {
          case BYLEVEL:
            processByLevelMessage(msg, session);
            break;
          case BYORDER:
            processByOrderMessage(msg, session);
            break;
        }
      }
    }
    return true;
}

/*-------------------------------------------------------------------------------------
 * Name        : processByOrderEvent
 * Description : process by order message
 * Arguments   : msg is the tick data message
 *             : session is the API session
```

```
       * Returns     : none
       *-------------------------------------------------------------------------------------*/
      void processByOrderMessage(const Message& msg, Session* session)
      {
        int side = -1;
        int position = -1;
        int bidRetran = 0;
        int askRetran = 0;

        // get gap detection flag (AMD book only)
        if (msg.hasElement(MD_GAP_DETECTED, true) && !d_gapDetected)
        {
          d_gapDetected = true;
          std::cout << "Bloomberg detected a gap in data stream." << std::endl;
        }

        // get event sub type
        Name subType = msg.getElement(MKTDEPTH_EVENT_SUBTYPE).getValueAsName();
        // get retran flags
        bidRetran = (subType == BID_RETRANS) ? 1 : 0;
        askRetran = (subType == ASK_RETRANS) ? 1 : 0;
        // BID/ASK message
        if (subType == BID || subType == ASK || bidRetran || askRetran)
        {
          if (subType == BID || bidRetran)
          {
            // bid side
            side = BIDSIDE;
          }
          else if (subType == ASK || askRetran)
          {
            // ask side
            side = ASKSIDE;
          }

          // get position
          int position = -1;
          if (msg.hasElement(POSITION_FIELD[BYORDER][side], true))
          {
            position = msg.getElement(POSITION_FIELD[BYORDER][side]).getValueAsInt32();
            if (position > 0) --position;
          }

          //  BID/ASK retran message
          if (askRetran || bidRetran)
          {
            // check for multi tick
            if (msg.hasElement(MD_MULTI_TICK_UPD_RT, true))
            {
              // multi tick
              if (msg.getElement(MD_MULTI_TICK_UPD_RT).getValueAsInt32() == 0 )
              {
                // last multi tick message, reset sequence number so next non-retran
                // message sequence number will be use as new starting number
                d_sequenceNumber = 0;
                if (askRetran && d askRetran)
                {
                  // end of ask retran
                  d_askRetran = false;
                  std::cout << "Ask retran completed." << std::endl;
                }
                else if (bidRetran && d bidRetran)
                {
                  // end of ask retran
                  d_bidRetran = false;
                  std::cout << "Bid retran completed." << std::endl;
                }
                if (!(d_askRetran || d_bidRetran))
```

```
            {
              // retran completed
              if (d_gapDetected)
              {
                // gap detected retran completed
                d_gapDetected = false;
                std::cout << "Gap detected retran completed." << std::endl;
              }
              else
              {
                // normal retran completed
                std::cout << "Retran completed." << std::endl;
              }
            }
          }
          else
          {
            if (askRetran && !d_askRetran)
            {
              // start of ask retran
              d_askRetran = true;
              std::cout << "Ask retran started." << std::endl;
            }
            else if (bidRetran && !d_bidRetran)
            {
              // start of ask retran
              d bidRetran = true;
              std::cout << "Bid retran started." << std::endl;
            }
          }
        }
      }
      else if (msg.hasElement(MBO SEQNUM RT, true))
      {
        // get sequence number
        long currentSequence = (long)msg.getElementAsInt64(MBO_SEQNUM_RT);
        if (d_sequenceNumber == 0 || d_sequenceNumber == 1 || (currentSequence == 1 &&
    d_sequenceNumber > 1))
        {
          // use current sequence number
          d_sequenceNumber = currentSequence;
        }
        else if ((d_sequenceNumber + 1 != currentSequence) && !d_gapDetected)
        {
          if (!d resubscribed)
          {
            // previous tick sequence can not be smaller than current tick
            // sequence number - 1 and NOT in gap detected mode.
            std::cout << "Warning: Gap detected - previous sequence number is "
                      << d sequenceNumber << " and current tick sequence number is "
                      << currentSequence << ")." << std::endl;
            // gap detected, re-subscribe to securities
            session->resubscribe(d_subscriptions);
            d_resubscribed = true;
          }
        }
        else if (d_sequenceNumber >= currentSequence)
        {
          // previous tick sequence number can not be greater or equal
          // to current sequence number
          std::cout << "Warning: Current Sequence number (" << currentSequence
                    << ") is smaller or equal to previous tick sequence number ("
                    << d_sequenceNumber << ")." << std::endl;
        }
        else
        {
          // save current sequence number
          d_sequenceNumber = currentSequence;
```

```
        }
      }

      // get command
      Name cmd = msg.getElement(MD_TABLE_CMD_RT).getValueAsName();
      if (cmd == CLEARALL)
      {
        d orderBooks[side].doClearAll();
      }
      else if (cmd == DEL)
      {
        d_orderBooks[side].doDel(position);
      }
      else if (cmd == DELALL)
      {
        d orderBooks[side].doDelAll();
      }
      else if (cmd == DELBETTER)
      {
        d_orderBooks[side].doDelBetter(position);
      }
      else if (cmd == DELSIDE)
      {
        d_orderBooks[side].doDelSide();
      }
      else if (cmd == REPLACE_CLEAR)
      {
        d_orderBooks[side].doReplaceClear(position);
      }
      else
      {
        // process other data commands
        // get price
        double fPrice = msg.getElement(PRICE_FIELD[BYORDER][side]).getValueAsFloat64();
        // get size
        unsigned int nSize = 0;
        if (msg.hasElement(SIZE_FIELD[BYORDER][side], true))
        {
          nSize = (unsigned int)msg.getElement(SIZE FIELD[BYORDER][side]).getValueAsInt64();
        }
        // get broker
        std::string sBroker = "";
        if (msg.hasElement(BROKER_FIELD[BYORDER][side], true))
        {
          sBroker = msg.getElement(BROKER FIELD[BYORDER][side]).getValueAsString();
        }
        // get time
        Datetime timeStamp = msg.getElement(TIME_FIELD[BYORDER]).getValueAsDatetime();
        std::stringstream ssTime;
        ssTime << setfill('0') << setw(2) << timeStamp.hours()
               << ":" << setfill('0') << setw(2) << timeStamp.minutes()
               << ":" << setfill('0') << setw(2) << timeStamp.seconds()
               << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
        // create entry
        ByOrderBookEntry entry(sBroker, (float)fPrice, ssTime.str(), 0, nSize);

        // process data command
        if (cmd == ADD) d orderBooks[side].doAdd(position, entry);
        else if (cmd == MOD) d_orderBooks[side].doMod(position, entry);
        else if (cmd == REPLACE) d_orderBooks[side].doReplace(position, entry);
        else if (cmd == REPLACE_BY_BROKER) d_orderBooks[side].doReplaceByBroker(entry);
        else if (cmd == EXEC) d orderBooks[side].doExec(position, entry);
      }
    }
    else
    {
      if (subType == TABLE_INITPAINT)
      {
```

```
      if (msg.fragmentType() == Message::FRAGMENT_START || msg.fragmentType() ==
Message::FRAGMENT NONE)
      {
        // init paint
        if (msg.hasElement(MBO_WINDOW_SIZE, true))
        {
          d_orderBooks[ASKSIDE].window_size = (unsigned int)
msg.getElementAsInt64(MBO WINDOW SIZE);
          d_orderBooks[BIDSIDE].window_size = d_orderBooks[ASKSIDE].window_size;
        }
        d_orderBooks[ASKSIDE].book_type = msg.getElementAsString(MD_BOOK_TYPE);
        d_orderBooks[BIDSIDE].book_type = d_orderBooks[ASKSIDE].book_type;
        // clear cache
        d orderBooks[ASKSIDE].doClearAll();
        d_orderBooks[BIDSIDE].doClearAll();
      }

      // ASK table
      Element askTable;
      if ((msg.asElement().getElement(&askTable, MBO_TABLE_ASK) == 0) && !askTable.isNull())
      {
        // has ask table array
        size_t numOfItems = askTable.numValues();
        for (size_t index = 0; index < numOfItems; ++index)
        {
          Element ask = askTable.getValueAsElement(index);
          // get command
          Name cmd = ask.getElement(MD_TABLE_CMD_RT).getValueAsName();
          // get position
          int position = -1;
          if (ask.hasElement(POSITION_FIELD[BYORDER][ASKSIDE], true))
          {
            position = ask.getElement(POSITION FIELD[BYORDER][ASKSIDE]).getValueAsInt32();
            if (position > 0) --position;
          }
          // get price
          double askPrice =
ask.getElement(PRICE_FIELD[BYORDER][ASKSIDE]).getValueAsFloat64();
          // get size
          unsigned int askSize = 0;
          if (ask.hasElement(SIZE_FIELD[BYORDER][ASKSIDE], true))
          {
            askSize = (unsigned
int)ask.getElement(SIZE_FIELD[BYORDER][ASKSIDE]).getValueAsInt64();
          }
          // get broker
          std::string askBroker = "";
          if (ask.hasElement(BROKER_FIELD[BYORDER][ASKSIDE], true))
          {
            askBroker = ask.getElement(BROKER FIELD[BYORDER][ASKSIDE]).getValueAsString();
          }
          // get time
          Datetime timeStamp = ask.getElement(TIME_FIELD[BYORDER]).getValueAsDatetime();
          std::stringstream askTime;
          askTime << setfill('0') << setw(2) << timeStamp.hours()
                  << ":" << setfill('0') << setw(2) << timeStamp.minutes()
                  << ":" << setfill('0') << setw(2) << timeStamp.seconds()
                  << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
          // create entry
          ByOrderBookEntry entry(askBroker, (float)askPrice, askTime.str(), 0, askSize);

          // process data command
          if (cmd == ADD) d_orderBooks[ASKSIDE].doAdd(position, entry);
          else if (cmd == MOD) d orderBooks[ASKSIDE].doMod(position, entry);
          else if (cmd == REPLACE) d_orderBooks[ASKSIDE].doReplace(position, entry);
          else if (cmd == REPLACE_BY_BROKER) d_orderBooks[ASKSIDE].doReplaceByBroker(entry);
          else if (cmd == EXEC) d_orderBooks[ASKSIDE].doExec(position, entry);
        }
```

```
        }
        // BID table
        Element bidTable;
        if ((msg.asElement().getElement(&bidTable, MBO TABLE BID) == 0) && !bidTable.isNull())
        {
          // has bid table array
          size_t numOfItems = bidTable.numValues();
          for (size t index = 0; index < numOfItems; ++index)
          {
            Element bid = bidTable.getValueAsElement(index);
            // get command
            Name cmd = bid.getElement(MD_TABLE_CMD_RT).getValueAsName();
            // get position
            int position = -1;
            if (bid.hasElement(POSITION_FIELD[BYORDER][BIDSIDE], true))
            {
              position = bid.getElement(POSITION_FIELD[BYORDER][BIDSIDE]).getValueAsInt32();
              if (position > 0) --position;
            }
            // get price
            double                                  bidPrice                                    =
    bid.getElement(PRICE FIELD[BYORDER][BIDSIDE]).getValueAsFloat64();
            // get size
            unsigned int bidSize = 0;
            if (bid.hasElement(SIZE FIELD[BYORDER][BIDSIDE], true))
            {
              bidSize                                   =                                   (unsigned
    int)bid.getElement(SIZE_FIELD[BYORDER][BIDSIDE]).getValueAsInt64();
            }
            // get broker
            std::string bidBroker = "";
            if (bid.hasElement(BROKER_FIELD[BYORDER][BIDSIDE], true))
            {
              bidBroker = bid.getElement(BROKER_FIELD[BYORDER][BIDSIDE]).getValueAsString();
            }
            // get time
            Datetime timeStamp = bid.getElement(TIME_FIELD[BYORDER]).getValueAsDatetime();
            std::stringstream bidTime;
            bidTime << setfill('0') << setw(2) << timeStamp.hours()
                    << ":" << setfill('0') << setw(2) << timeStamp.minutes()
                    << ":" << setfill('0') << setw(2) << timeStamp.seconds()
                    << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
            // create entry
            ByOrderBookEntry entry(bidBroker, (float)bidPrice, bidTime.str(), 0, bidSize);

            // process data command
            if (cmd == ADD) d_orderBooks[BIDSIDE].doAdd(position, entry);
            else if (cmd == MOD) d_orderBooks[BIDSIDE].doMod(position, entry);
            else if (cmd == REPLACE) d_orderBooks[BIDSIDE].doReplace(position, entry);
            else if (cmd == REPLACE BY BROKER) d orderBooks[BIDSIDE].doReplaceByBroker(entry);
            else if (cmd == EXEC) d_orderBooks[BIDSIDE].doExec(position, entry);
          }
        }
      }
    }
    return;
  }

  /*--------------------------------------------------------------------------------
   * Name      : processByLevelEvent
   * Description  : process by level message
   * Arguments  : msg is the tick data message
   *              : session is the API session
   * Returns    : none
   *-------------------------------------------------------------------------------*/
  void processByLevelMessage(const Message& msg, Session* session)
  {
    int side = -1;
```

```
        int position = -1;
        int bidRetran = 0;
        int askRetran = 0;

        // get gap detection flag (AMD book only)
        if (msg.hasElement(MD_GAP_DETECTED, true) && !d_gapDetected)
        {
          d gapDetected = true;
          std::cout << "Bloomberg detected a gap in data stream." << std::endl;
        }

        // get event subtype
        Name subType = msg.getElement(MKTDEPTH_EVENT_SUBTYPE).getValueAsName();
        // get retran flags
        bidRetran = (subType == BID_RETRANS) ? 1 : 0;
        askRetran = (subType == ASK RETRANS) ? 1 : 0;
        // BID or ASK message
        if (subType == BID || subType == ASK || bidRetran || askRetran)
        {
          // set book size
          if(subType == BID || bidRetran)
          {
            side = BIDSIDE;
          }
          else if (subType == ASK || askRetran)
          {
            side = ASKSIDE;
          }

          // get position
          int position = -1;
          if (msg.hasElement(POSITION_FIELD[BYLEVEL][side], true))
          {
            position = msg.getElement(POSITION_FIELD[BYLEVEL][side]).getValueAsInt32();
            if (position > 0) --position;
          }

          //  BID/ASK retran message
          if (askRetran || bidRetran)
          {
            // check for multi tick
            if (msg.hasElement(MD_MULTI_TICK_UPD_RT, true))
            {
              // multi tick
              if (msg.getElement(MD MULTI TICK UPD RT).getValueAsInt32() == 0 )
              {
                // last multi tick message, reset sequence number so next non-retran
                // message sequence number will be use as new starting number
                d_sequenceNumber = 0;
                if (askRetran && d askRetran)
                {
                  // end of ask retran
                  d_askRetran = false;
                  std::cout << "Ask retran completed." << std::endl;
                }
                else if (bidRetran && d bidRetran)
                {
                  // end of ask retran
                  d_bidRetran = false;
                  std::cout << "Bid retran completed." << std::endl;
                }
                if (!(d askRetran || d bidRetran))
                {
                  // retran completed
                  if (d_gapDetected)
                  {
                    // gap detected retran completed
                    d_gapDetected = false;
```

```
                std::cout << "Gap detected retran completed." << std::endl;
              }
              else
              {
                // normal retran completed
                std::cout << "Retran completed." << std::endl;
              }
            }
          }
          else
          {
            if (askRetran && !d_askRetran)
            {
              // start of ask retran
              d_askRetran = true;
              std::cout << "Ask retran started." << std::endl;
            }
            else if (bidRetran && !d_bidRetran)
            {
              // start of ask retran
              d_bidRetran = true;
              std::cout << "Bid retran started." << std::endl;
            }
          }
        }
      }
      else if (msg.hasElement(MBA SEQNUM RT, true))
      {
        // get sequence number
        long currentSequence = (long)msg.getElementAsInt64(MBA_SEQNUM_RT);
        if (d_sequenceNumber == 0 || d_sequenceNumber == 1 || (currentSequence == 1 &&
  d_sequenceNumber > 1))
        {
          // use current sequence number
          d_sequenceNumber = currentSequence;
        }
        else if ((d_sequenceNumber + 1 != currentSequence) && !d_gapDetected)
        {
          if (!d resubscribed)
          {
            // previous tick sequence can not be smaller than current tick
            // sequence number - 1 and NOT in gap detected mode.
            std::cout << "Warning: Gap detected - previous sequence number is "
                      << d_sequenceNumber << " and current tick sequence number is "
                      << currentSequence << ")." << std::endl;
            // gap detected, re-subscribe to securities
            session->resubscribe(d_subscriptions);
            d_resubscribed = true;
          }
        }
        else if (d_sequenceNumber >= currentSequence)
        {
          // previous tick sequence number can not be greater or equal
          // to current sequence number
          std::cout << "Warning: Current Sequence number (" << currentSequence
                    << ") is smaller or equal to previous tick sequence number ("
                    << d_sequenceNumber << ")." << std::endl;
        }
        else
        {
          // save current sequence number
          d sequenceNumber = currentSequence;
        }
      }

      // get command
      Name cmd = msg.getElement(MD_TABLE_CMD_RT).getValueAsName();
      if (cmd == CLEARALL)
```

```
        {
          d levelBooks[side].doClearAll();
        }
        else if (cmd == DEL)
        {
          if (position != -1) d_levelBooks[side].doDel(position - 1);
        }
        else if (cmd == DELALL)
        {
          d levelBooks[side].doDelAll();
        }
        else if (cmd == DELBETTER)
        {
          d levelBooks[side].doDelBetter(position - 1);
        }
        else if (cmd == DELSIDE)
        {
          d_levelBooks[side].doDelSide();
        }
        else if (cmd == REPLACE_CLEAR)
        {
          d levelBooks[side].doReplaceClear(position - 1);
        }
        else
        {
          // process other commands
          // get price
          double fPrice = msg.getElement(PRICE_FIELD[BYLEVEL][side]).getValueAsFloat64();
          // get size
          unsigned int nSize = 0;
          if (msg.hasElement(SIZE_FIELD[BYLEVEL][side], true))
          {
            nSize = (unsigned int)msg.getElement(SIZE FIELD[BYLEVEL][side]).getValueAsInt64();
          }
          // get number of order
          unsigned int nNumOrder = 0;
          if (msg.hasElement(ORDER_FIELD[BYLEVEL][side], true))
          {
            nNumOrder                                    =                                    (unsigned
    int)msg.getElement(ORDER_FIELD[BYLEVEL][side]).getValueAsInt64();
          }
          // get time
          Datetime timeStamp = msg.getElement(TIME_FIELD[BYLEVEL]).getValueAsDatetime();
          std::stringstream ssTime;
          ssTime << setfill('0') << setw(2) << timeStamp.hours()
                 << ":" << setfill('0') << setw(2) << timeStamp.minutes()
                 << ":" << setfill('0') << setw(2) << timeStamp.seconds()
                 << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
          // create entry
          ByLevelBookEntry entry((float)fPrice, ssTime.str(), nNumOrder, nSize);

          // process data command
          if (cmd == ADD) d_levelBooks[side].doAdd(position, entry);
          else if (cmd == MOD) d_levelBooks[side].doMod(position, entry);
          else if (cmd == REPLACE) d_levelBooks[side].doReplace(position, entry);
          else if (cmd == EXEC) d levelBooks[side].doExec(position, entry);
        }
      }
      else
      {
        if (subType == TABLE_INITPAINT)
        {
          if   (msg.fragmentType()   ==   Message::FRAGMENT_START   ||   msg.fragmentType()   ==
    Message::FRAGMENT NONE)
          {
            // init paint
            if (msg.hasElement(MBA_WINDOW_SIZE, true))
            {
```

```
            d_levelBooks[ASKSIDE].window_size           =           (unsigned           int)
    msg.getElementAsInt64(MBA WINDOW SIZE);
            d_levelBooks[BIDSIDE].window_size = d_levelBooks[ASKSIDE].window_size;
          }
          d_levelBooks[ASKSIDE].book_type = msg.getElementAsString(MD_BOOK_TYPE);
          d_levelBooks[BIDSIDE].book_type = d_levelBooks[ASKSIDE].book_type;
          // clear cache
          d levelBooks[ASKSIDE].doClearAll();
          d_levelBooks[BIDSIDE].doClearAll();
        }

        // ASK table
        Element askTable;
        if ((msg.asElement().getElement(&askTable, MBA TABLE ASK) == 0) && !askTable.isNull())
        {
          // has ask table array
          size_t numOfItems = askTable.numValues();
          for (size_t index = 0; index < numOfItems; ++index)
          {
            Element ask = askTable.getValueAsElement(index);
            // get command
            Name cmd = ask.getElement(MD TABLE CMD RT).getValueAsName();
            // get position
            position = -1;
            if (ask.hasElement(POSITION FIELD[BYLEVEL][ASKSIDE], true))
            {
              position = ask.getElement(POSITION FIELD[BYLEVEL][ASKSIDE]).getValueAsInt32();
              if (position > 0) --position;
            }
            // get price
            double                                  askPrice                                    =
    ask.getElement(PRICE_FIELD[BYLEVEL][ASKSIDE]).getValueAsFloat64();
            // get size
            unsigned int askSize = 0;
            if (ask.hasElement(SIZE_FIELD[BYLEVEL][ASKSIDE], true))
            {
              askSize                                  =                                (unsigned
    int)ask.getElement(SIZE_FIELD[BYLEVEL][ASKSIDE]).getValueAsInt64();
            }
            // get number of order
            unsigned int askNumOrder = 0;
            if (ask.hasElement(ORDER_FIELD[BYLEVEL][ASKSIDE], true))
            {
              askNumOrder                                =                                (unsigned
    int)ask.getElement(ORDER FIELD[BYLEVEL][ASKSIDE]).getValueAsInt64();
            }
            // get time
            Datetime timeStamp = ask.getElement(TIME_FIELD[BYLEVEL]).getValueAsDatetime();
            std::stringstream askTime;
            askTime << setfill('0') << setw(2) << timeStamp.hours()
                    << ":" << setfill('0') << setw(2) << timeStamp.minutes()
                    << ":" << setfill('0') << setw(2) << timeStamp.seconds()
                    << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
            // create entry
            ByLevelBookEntry entry((float)askPrice, askTime.str(), askNumOrder, askSize);

            // process data command
            if (cmd == ADD) d levelBooks[ASKSIDE].doAdd(position, entry);
            else if (cmd == MOD) d_levelBooks[ASKSIDE].doMod(position, entry);
            else if (cmd == REPLACE) d_levelBooks[ASKSIDE].doReplace(position, entry);
            else if (cmd == EXEC) d_levelBooks[ASKSIDE].doExec(position, entry);
          }
        }
        // BID table
        Element bidTable;
        if ((msg.asElement().getElement(&bidTable, MBA_TABLE_BID) == 0) && !bidTable.isNull())
        {
          // has bid table array
```

```
            size_t numOfItems = bidTable.numValues();
            for (size_t index = 0; index < numOfItems; ++index)
            {
              Element bid = bidTable.getValueAsElement(index);
              // get command
              Name cmd = bid.getElement(MD_TABLE_CMD_RT).getValueAsName();
              // get position
              int position = -1;
              if (bid.hasElement(POSITION_FIELD[BYLEVEL][BIDSIDE], true))
              {
                position = bid.getElement(POSITION_FIELD[BYLEVEL][BIDSIDE]).getValueAsInt32();
                if (position > 0) --position;
              }
              // get price
              double                                    bidPrice                                    =
    bid.getElement(PRICE FIELD[BYLEVEL][BIDSIDE]).getValueAsFloat64();
              // get size
              unsigned int bidSize = 0;
              if (bid.hasElement(SIZE_FIELD[BYLEVEL][BIDSIDE], true))
              {
                bidSize                                   =                                   (unsigned
    int)bid.getElement(SIZE FIELD[BYLEVEL][BIDSIDE]).getValueAsInt64();
              }
              // get number of order
              unsigned int bidNumOrder = 0;
              if (bid.hasElement(ORDER_FIELD[BYLEVEL][BIDSIDE], true))
              {
                bidNumOrder                               =                                   (unsigned
    int)bid.getElement(ORDER_FIELD[BYLEVEL][BIDSIDE]).getValueAsInt64();
              }
              // get time
              Datetime timeStamp = bid.getElement(TIME_FIELD[BYLEVEL]).getValueAsDatetime();
              std::stringstream bidTime;
              bidTime << setfill('0') << setw(2) << timeStamp.hours()
                      << ":" << setfill('0') << setw(2) << timeStamp.minutes()
                      << ":" << setfill('0') << setw(2) << timeStamp.seconds()
                      << "." << setfill('0') << setw(3) << timeStamp.milliSeconds();
              // create entry
              ByLevelBookEntry entry((float)bidPrice, bidTime.str(), bidNumOrder, bidSize);

              // process data command
              if (cmd == ADD) d_levelBooks[BIDSIDE].doAdd(position, entry);
              else if (cmd == MOD) d_levelBooks[BIDSIDE].doMod(position, entry);
              else if (cmd == REPLACE) d_levelBooks[BIDSIDE].doReplace(position, entry);
              else if (cmd == EXEC) d levelBooks[BIDSIDE].doExec(position, entry);
            }
          }
        }
      }
      return;
    }
```

11.1.2 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 to be 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.

11.1.3 TYPES OF ORDER BOOKS

There are three types of book; Market-By-Order (MBO), Market-By-Level (MBL) and Market Maker Quote (MMQ). An Exchange operating an order book could provide MBO only, MBL only or both. In some cases, the Exchange provides an MBO book, with the MBL book being derived by Bloomberg. It is possible to aggregate an MBO into an MBL book, but an MBL book cannot be split into its component orders. An Exchange operating a quote book would provide MMQ. In some rare instances, a given security may support both an order book (MBO and/or MBL) and a quote book (MMQ) if the market supports both trading mechanisms on the same security. An example of such a market is the SETSqx market at London Stock Exchange.

11.1.3.1 MARKET-BY-ORDER (MBO)

An MBO book provides every order in the book, subject to the constraints defined by the view and window-size attributes. If multiple brokers have orders at the same price level, the book will show each order — resulting in multiple rows sharing the same price. The amount of data available at each level varies by the source of the data, but it typically consists of the price, size, time of the order and, in some instances, a broker ID. Positions are amended or removed from MBO books as orders are matched and partially or completely executed on the exchange.

11.1.3.2 MARKET-BY-LEVEL (MBL)

An MBL order book is the aggregated market-by-price/yield (previously often called Market-By-Level). This displays only one position (row) for each unique price. If multiple brokers have the same price, then the size of all of their orders will be accumulated and displayed against that price.

As orders are matched and executed at the exchange, the volume available at a price may be completely or partially consumed and updates are provided so clients can represent the available price and volume as market conditions change.

11.1.3.3 MARKET MAKER QUOTE (MMQ)

An MMQ book provides a collection of all the competing quotes from each of the brokers or market makers on a security. There are usually only two quotes (one best bid offer quote and one best ask offer quote) from each participant, commonly referred to as two-way quotes; these represent the prices at which that participant is obliged to buy or sell during a mandatory quotation period (hence they “make the market”). All participants compete against one another, and it is possible to rank the quotes in the MMQ book and thus build a virtual aggregated price book.

11.1.3.4 TOP BROKERS (TOP)

Top brokers is primarily used for the Hong Kong Exchange (HKEx) to provide the top 40 broker orders on each side of the market, but with prices only (no volumes).

11.1.4 ORDER BOOK METHODS

11.1.4.1 REPLACE-BY-POSITION (RBP)

In replace-by-position book management, the specific set of columns (size, number of orders, time, etc.) varies by Exchange. Often, an update to one level in an RBP book will cause changes to other levels. When many levels are updated as part of the same Event, the Multi-tick Update (MTU) flag may be included (if the MTU attribute value is equal to “ON”) so that clients will know when all updates are complete and the book is returned to a valid state.

This approach can be used for both MBO and MBL types of books. The updated methodology is straightforward. Clients should locate the position specified and overwrite the price, size and time at that position with the new data supplied.

11.1.4.2 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 up all the prices below it. 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.

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. Therefore, 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 (resubscribe to the book). If the gap is detected as a result of an issue within the Bloomberg Data Center, Bloomberg will send an order recap. This form of gap detection is covered in a later section.

11.1.4.3 REPLACE-BY-BROKER (RBB)

In replace-by-broker books, the bid/ask for a specific broker is communicated as a replacement of the bid/ask data that had previously been held for that broker.

This style is used solely for MMQ book types and it is a mixture of RBP and AMD update types; the book is built from broker entries and it is similar to the RBP Message in that rows are directly indexed (by row in RBP and by broker code in RBB).

How RBB order books are sorted is left up to the consuming application. The general rule is to follow price > time > size priority.

11.1.5 SUBSCRIBING TO MARKET DEPTH

The first step in subscribing to the `//blp/mktdepthdata` service is to learn how the Subscription strings are formulated. For the string to be valid, users must specify a "type" parameter, which can be either MBO (Market-by-Order) or MBL (Market-by-Level). Users 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
Ticker	//blp/mktdepthdata/ticker/symbol	//blp/mktdepthdata/**ticker**/VOD LN Equity?type=**MBO**
ISIN	//blp/mktdepthdata/isin/identifier source	//blp/mktdepthdata/**isin**/DE0005557508 TQ?type=**MBL**
CUSIP	//blp/mktdepthdata/cusip/identifier source	//blp/mktdepthdata/**cusip**/459200101 LN?type=**MBL**
SEDOL	//blp/mktdepthdata/sedol/identifier source	//blp/mktdepthdata/**sedol**/0540528 TQ?type=**MBL**
Bloomberg Unique ID	//blp/mktdepthdata/buid/identifier source	//blp/mktdepthdata/**buid**/EQ0000000000496862 JT?type=**MBL**
BSID	//blp/mktdepthdata/bsid/bsid	//blp/mktdepthdata/**bsid**/20057504821 38?type=**MBL**
ID_BB_Global	//blp/mktdepthdata/bbgid/bbgid /bbgid source	//blp/mktdepthdata/**bbgid**/BBG000BDQGR5 IX?type=**MBL**
CATS	//blp/mktdepthdata/cats/identifier source	//blp/mktdepthdata/**cats**/6888 MK?type=**MBL**
CINS	//blp/mktdepthdata/cins/identifier source	//blp/mktdepthdata/**cins**/G0408V102 US?type=**MBO**
COMMON	//blp/mktdepthdata/common/identifier source	//blp/mktdepthdata/**common**/025929551 LN?type=**MBO**
SICOVAM	//blp/mktdepthdata/sicovam/identifier source	//blp/mktdepthdata/**sicovam**/013000 FP?type=**MBL**
SVM	//blp/mktdepthdata/svm/identifier source	//blp/mktdepthdata/**svm**/356573 BB?type=**MBL**
WERTPAPIER	//blp/mktdepthdata/wpk/identifier source	//blp/mktdepthdata/**wpk**/803200 GY?type=**MBL**
AUSTRIA	//blp/mktdepthdata/austria/identifier source	//blp/mktdepthdata/**AUSTRIA**/080905 AV?type=**MBL**
BELG	//blp/mktdepthdata/belg/identifier source	//blp/mktdepthdata/**BELG**/381027 BB?type=**MBL**
Bloomberg Symbol	//blp/mktdepthdata/bsym/source/symbol	//blp/mktdepthdata/**bsym**/LN/VOD?type=**MBL** //blp/mktdepthdata/**bsym**/US/AAPL?type=**MBO**
Parsekeyable	//blp/mktdepthdata/bpkbl/bpkbl	//blp/mktdepthdata/**bpkbl**/QCZ1 Index?type=**MBL**
FRENCH	//blp/mktdepthdata/french/identifier source	//blp/mktdepthdata/**french**/013000 FP?type=**MBL**
IRISH	//blp/mktdepthdata/irish/identifier source	//blp/mktdepthdata/**IRISH**/3070732 ID?type=**MBL**
VALOREN	//blp/mktdepthdata/valoren/identifier source	//blp/mktdepthdata/**VALOREN**/002489948 VX?type=**MBL**

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

```
const char *security =
"//blp/mktdepthdata/isin/US/US4592001014?type=MBL"; SubscriptionList
subscriptions;

subscriptions.add(security, CorrelationId((char
*)security)); session.susbcribe (subscriptions);
```

11.1.6 RESPONSE OVERVIEW

The market-depth response will be a series of SUBSCRIPTION_DATA Events that users will already be familiar with if they 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; within each message 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_SUBTYP	Notes
TABLE_INITPAINT	This is the Initial Paint message for the Subscription. When this Message is received, it is an indicator to the user 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 the user should assign a unique CorrelationID to each one of their Subscriptions in order to map the Message updates to the initial request. For AMD and RBP book types, there will be a WINDOW_SIZE field/ value pairing, which indicates the number of levels in the book (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, Bloomberg infrastructure will automatically recover (RECAP) and send BID_RETRANS and ASK_RETRANS Events. Upon receipt of these Messages, user will receive a CLEARALL Message with a MKTDEPTH_EVENT_SUBTYPE of RETRANS; user should consider its 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, users 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, users will see on MD_TABLE_CMD field/value pairing for each BID or ASK MKTDEPTH_EVENT_SUBTYPE tick update.

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

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 this order is added in the market-depth table, users 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 users 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. The user is responsible for caching 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 positions 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, then 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, whereas 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 users 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 its 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 order book for the specified side. This market-depth table command is issued by Bloomberg when market-depth recovery is under way. 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, users will generally see 2 CLEARALLs — 1 for Bid side and 1 for Ask side. Should the client of market depth need to process a recovery of market depth differently, this table command allows the user to differentiate from the source/exchange produced DELETEALL. CLEARALL messages may occur without accompanying RETRANS labels in the event of data loss within Bloomberg network or on the receipt of the first tick of a new trading day. Hence, on receipt of a CLEARALL, users should clear their book and prepare to receive the subsequent recover ADD messages.
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 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 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 refer to the aforementioned “MarketDepthSubscriptionSnaphotExample” example in either the Java, C++ or .NET SDK.

```
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;

}
```

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. Also notice that the event handler for the tick updates is identical to that of a `//blp/mktdata` Subscription, for instance.

11.1.7 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. The user has the responsibility of handling this in its application.

Users will achieve the handling of multiple fragments 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 holds 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 two 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].doC
```

```
            learAll();
            d_orderBooks[Side::BIDSIDE].doC
            learAll();

        }

        }
```

The above code checks the market-depth Event sub-type being returned; 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.

11.1.8 DATA RESPONSE FOR ADD-MOD-DEL (AMD) ORDER BOOKS

Every Event in an Add-Mode-Delete (AMD) order book is critical to maintaining an accurate book. One missed Message can result in a book that is wrong for the remainder of the trading day. Accordingly, 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. The Sequence number is incremented per book. It is up to the user's application to clear the book as soon as it receives an Initial Paint Message.

11.1.8.1 MBO-AMD SAMPLE SUBSCRIPTION OUTPUT

(for "`//blp/mktdepthdata/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[] = {
            }
        }
```

```
        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 = "
                                        8
                    0" 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, users

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 users receive an Initial Paint Message, they should clear their 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.

11.1.9 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 RBP, an RBB order book has no concept of numbers. 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 the user's application to clear the book as soon as it receives an Initial Paint Message.

11.1.9.1 MBO-RBB SUBSCRIPTION OUTPUT

(for "`//blp/mktdepthdata/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, users 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 users receive an Initial Paint message, they should clear their 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.

11.1.10 DATA Response FOR REQUEST-BY-POSITION (RBP) ORDER BOOKS

With the Replace-By-Position (RBP) method, each level is explicitly sent so 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 good, have no impact other than that they were missed. All other levels retain their correct values.

The RBP method is generally easier to implement than AMD, but 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.

11.1.10.1 MBL-RBP SUBSCRIPTION OUTPUT

(for "`//blp/mktdepthdata/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-RBP
    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
        MBLASK_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:

MKTDEPTH_EVENT_TYPE
Within the Initial Paint Message, users 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.

MD_BOOK_TYPE WITH A VALUE OF "MBL-RBP".
Because this is a Request-By-Position (RBP) MBL book type, the MD_TABLE_CMD_RT field in the Initial Paint is "REPLACE" and all subsequent updates will have 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.

11.1.11 ORDER BOOK RECAPS

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

Recaps apply to every style of order book: Add-Mod-Delete (AMD), Replace-by-Position (RBP) 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. RBP and RBB books tend to be self-correcting in the event of a data gap, thus 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.

11.1.12 GAP DETECTION

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

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

MD_GAP_DETECTED will not be present once the recap is sent. Therefore, even though a client application detects a gap, if this field is present in market-depth update Messages, no further action is required by the

client application except to begin reading the recap Messages, which will follow immediately and be indicated with a MKTDEPTH_EVENT_SUBTYPE of BID_RETRANS and ASK_RETRANS in each Message update. In cases where a sequence number gap is detected but the MD_GAP_DETECTED field is not present in the Message, the client application is responsible for requesting a recap (i.e., resubscribe) to the order book.

FIELDS AFFECTED BY RECAPS

Fields	Descriptions
MKTDEPTH_EVENT_SUBTYPE	Present in every market-depth Message for all styles of order book. 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 that 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.

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 RBP style. order book updates.

11.2 MARKET LIST SERVICE (//BLP/MKTLIST)

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 NONBPS 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". The table below provides further details.

MARKET LIST STRING DEFINITIONS

<service owner>	For 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 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 below for additional information and an example.
<topic type>	See: Security Nomenclature	

11.2.1 CODE EXAMPLES

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

MARKETLISTSUBSCRIPTIONEXAMPLE
This example demonstrates how to make a simple Market List "chain" Subscription for one, or more, securities and displays all of the Messages to the console window.

MARKETLISTSNAPSHOTEXAMPLE
This example demonstrates how to make a Market List "secids" snapshot Request and displays the Message to the console window.

Now that users have a better understanding about how a //blp/mktlist Subscription or snapshot string is formed, it is time to use it in their application. The following sections provide further details about how to subscribe to a chain of instruments and request a snapshot of a list of members.

11.2.2 SUBSCRIBING TO INSTRUMENT CHAINS

OVERVIEW
B-PIPE supports the ability to subscribe to lists of instruments known as "chains". When a Subscription is made for a chain, the Request must first resolve to a single B-PIPE instrument. This instrument is called the "underlying instrument".

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

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

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

For most chains, the relationship between the underlying instrument and the list members is established by the B-PIPE service when the Subscription is made using the BSID of the underlying instrument. Every member of the list has a LIST_UNDERLYING_ID_BSID field, which contains the BSID value of the underlying instrument and all matching instruments of the appropriate security class are returned in the list of members.

Index and Curve lists are handled differently. The list's members are maintained by the Bloomberg Data Center. Once it is determined that this list Subscription is for Index or Curve members, the Bloomberg Data Center is queried for the list of members. This list contains the Terminal Ticker (Parsekeyable symbol) for each member, which is resolved to an instrument on B-PIPE. It is possible that an Index or Curve list member is not available on B-PIPE. In this case, the list member will be included in the list, but return only the Parsekeyable symbol.

This allows the requestor to contact Bloomberg about getting the missing instrument added to B-PIPE.

The default security class of the list members depends on the security class of the underlying instrument specified in the Request. The default can be overridden using the optional parameter "secclass". The table below defines the default security class of the list members for each underlying instrument security class.

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, a list of futures with IBM can be obtained as the underlying instrument:

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

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 number of returned members to those belonging to the specified source(s) only. More than one value is allowed for this parameter.

The "source" can be substituted by a "~". This value can be used when the client assumes that there is only one source for the security and there is no actual need to specify it. If this is the case, the Subscription request will be processed successfully, but if the security has more than one source and the request is ambiguous, then the client will receive a SubscriptionFailure response with a NOTUNIQUE description. An example of such a Subscription string would be "`//blp/mktlist/chain/cusip/~/459200101`".

11.2.3 CHAIN SUBSERVICE EXAMPLES

Type of Chain List	Example Subscription String	Topic Type	Topic Key[a]	Re-fresh[b]
Option Chains	`//blp/mktlist/chain/bsym/LN/VOD`	`/bsym`	/<DX282>/<DY003>	No
	`//blp/mktlist/chain/bsid/678605358297`	`/bsid`	/<ID122>	No
	`//blp/mktlist/chain/buid/LN/EQ0010160500001`	`/buid`	/<DX282>/<ID059>	No
	`//blp/mktlist/chain/bbid/LN/EQ0010160500001`	`/bbid`	/<DX282>/<ID059>	No
	`//blp/mktlist/chain/bpkbl/VOD LN Equity`	`/bpkbl`	/<DX194>	No
	`//blp/mktlist/chain/cusip/UN/459200101`	`/cusip`	/<DX282>/<ID032>	No
	`//blp/mktlist/chain/isin/LN/GB00BH4HKS39`	`/isin`	/<DX282>/<ID005>	No
	`//blp/mktlist/chain/sedol/LN/BH4HKS3`	`/sedol`	/<DX282>/<ID002>	No
	`//blp/mktlist/chain/bbgid/LN/BBG000C6K6G9`	`/bbgid`	/<DX282>/<ID135>	No
	`//blp/mktlist/chain/ticker/VOD LN Equity`	`/ticker`	/<DX194>	No
	`//blp/mktlist/chain/bsym/FTUK/UKX`	`/bsym`	/<DX282>/<DY003>	Daily
Index List	`//blp/mktlist/chain/bpkbl/YCMM0010 Index`	`/bpkbl`	/<identifier>	Daily
Yield Curve	`//blp/mktlist/chain/gdco/broker/id`	`/gdco`	/<broker_id>/<mon_id>	N/A
GDCO	`//blp/mktlist/chain/eid/14014`	`/eid`	/<source>	No
EID List	`//blp/mktlist/chain/source/UN;secclass=Equi`	`/source`	/<source>	No
Source List				

a. The FLDS <GO> identifier associated with the expected key values for that particular topic is listed, where applicable; it 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

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`	Returns options on the UKX Index traded on
`//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
`//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.

Subscription String	Returns
//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/bpkbl/IBM US Equity";
SubscriptionList subscriptions;
subscriptions.add(security, CorrelationId((char
*)security)); session.susbcribe (subscriptions);
```

11.2.4 RESPONSE OVERVIEW

The Market List response will be a series of SUBSCRIPTION_DATA Events that users will be familiar with if they have developed Bloomberg API applications using any of the other streaming services, such as //blp/mktdata, //blp/mktvwap or //blp/mktdepthdata.

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 the chain of instruments. Within a single ListRecap Message, users 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 the equity has no options, 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 ListData items could be received later on—so users 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 it will not return any further updates and users may wish to terminate the Subscription by unsubscribing. This is explained further below.

Various types of lists are available for Subscription. Although 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.

11.2.5 LIST ACTIONS

ListAction	Description
CLEAR	Delete all existing list members. This implies more data is 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 must match the current Member Identifiers exactly
END	The last set in the sequence.
CLEAR_AND_END	Delete all of the existing list members as no more entries will 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 must match the current Member Identifiers exactly. Then end as there are no more entries in this sequence.

11.2.6 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":

```
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
        }



        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) and a single ListData Message, which is an actual update to the Subscription. Although the ListRecap does not possess a LIST_ACTION value, users are to treat such a Message as a CLEAR_AND_ADD action. In other words, the user will clear its cache and add the entries included in the Message.

In the ListRecap Message, users will notice a few other pieces of information in addition to the entries, such as the LIST_LISTTYPE field (in this case, its value is “Source List”, which they 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 lists’ constituent instruments can change.

Following the ListRecap Message, users 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 the ADD will be at the END of the list (as indicated by ADD_AND_END).

11.2.7 HANDLING MULTIPLE MESSAGES (A.K.A. FRAGMENTS)

The summary (Initial Paint) Messages can be split into one or more smaller Messages if the returned data is too large to fit into a single Message. The user's application must handle this.

Users 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 occupies 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 two parts, all middle Fragments will be of type FRAGMENT_INTERMEDIATE. 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, users will call the fragmentType property of the Message object (e.g., msg.fragmentType()). Within their application, they will check to see if the Fragment type of the ListRecap Message is FRAGMENT_NONE or FRAGMENT_START. If one of these is determined, then users will want to clear their list and begin adding the entries included in that part of the ListRecap Message. In the case where FRAGMENT_START is determined, then they will know to continue reading the ListRecap Messages and adding the entries to their list from those Messages until they receive a ListRecap with a Fragment type for FRAGMENT_END. At this point, users are to indicate that they have finished building their list; it is now time to wait for any subsequent ListData updates.

11.2.8 SNAPSHOT REQUEST FOR LIST OF SECURITY IDENTIFIERS

If users want to retrieve a list of all available sources that are pricing a given instrument, then they 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. With this subservice, users also have the ability to filter their results to only a particular source.

The following table lists all of the supported topic types, their applicable topic key formats and associated B-PIPE mnemonic and FLDS <GO> field Identifiers.

Topic Type	Topic Key	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

Topic Type	Topic Key	B-PIPE Field	FLDS <GO> Field
/cusip	/<identifier>	ID_CUSIP	ID032
/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, thus the returned list of instruments does not receive update Messages and must be re-requested to discover any new pricing sources that have emerged since the initial request. Listed below are the 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/EQ001008010`	
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/US459200101`	
Parsekeyable	`//blp/mktlist/secids/bpkbl/parsekeyab`	All instrument IDs for the given `Parsekeyable`
	`//blp/mktlist/secids/bpkbl/UKX Index`	

Listed below are the 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 shows such an instrument with and without the “source” parameter. Listed below are the market list Requests with the secids subservice name:

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);
```

11.2.9 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 their application, users will handle the data response the same way, initially, as they would for any static request. This is accomplished by checking the Event type of the incoming Message. If its Event type is PARTIAL_RESPONSE, that indicates at least one more Message is to be received to fulfill that request. Users will continue reading the incoming Messages until they receive a RESPONSE Event type, which indicates that the Request has been fully served.

☞ *For additional information, refer to the "Reference Services and Schemas Guide".*

Below a sample Event handler written in C++. It was extracted from the "MarketListSnapshotExample" example found in the B-PIPE C++ API SDK and is the event handler responsible for displaying the above output to a console window.

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

```
// 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 users examine the response from the example market list request, which is "`//blp/mktlist/secids/cusip/459200101;source=US`", they will find the data all returned in a single Message; the Message will have an Event type of RESPONSE. Within that block of code is a call to processResponseEvent(). It is here that a check is made first for the responseCode Element. To understand the reason for checking for this Element, users will first need to understand the structure of the schema for the `//blp/mktlist` service. Displayed below is a screenshot capturing the sub-elements of the SnapshotRequest/Responses node.

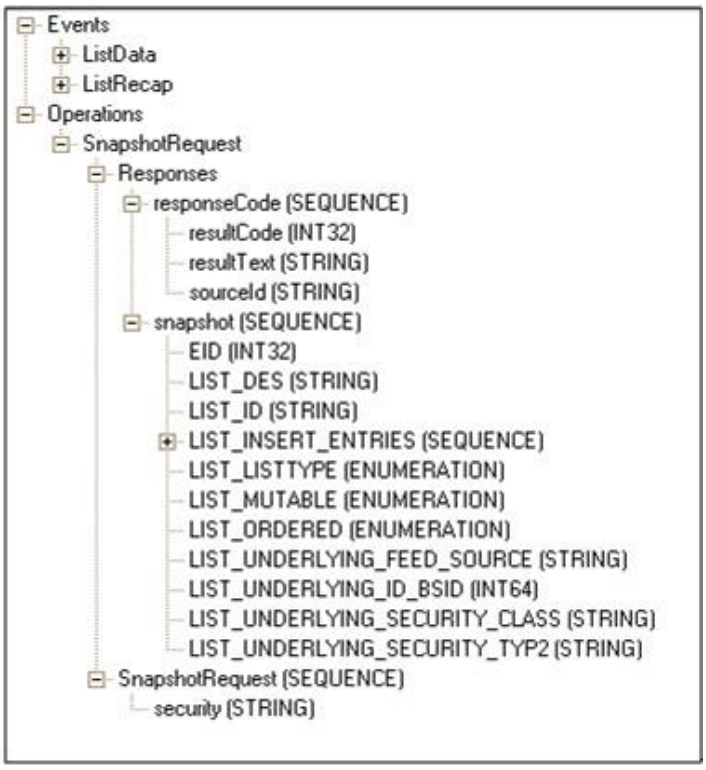


Figure 14. SnapshotRequest/Responses node

If the responseCode is found in the Message, then users check to see if the resultCode is greater than zero. If it is, itindicates a problem with the Request and that this Message contains an error. The details of the error are provided by the Message's resultCode, resultText and sourceId values.

If the resultCode equals zero, then the Message will contain data. In this case, the snapshot element of the Message is retrieved. In the above processResponseEvent() handler the number of Elements contained in the snapshot are determined by a call to numElements(); then each of those Elements is read into a dataItem variable, of type Element, one at a time. Users 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.

The schema screenshot shows a total of 10 possible single-field Elements and 1 array Element in a snapshot. The array Element is indicated by the SEQUENCE type. In this case, the resultCode is zero (i.e., no errors), with 6 Elements contained in the snapshot Element. The first 3 are single-field elements (e.g., LIST_ID, EID, LIST_LISTTYPE), so isArray() returns false for each of them. However, the fourth element, LIST_INSERT_ENTRIES, is an array (a.k.a. SEQUENCE type). This element is processed in the processBulkData() function. The remaining two Elements (LIST_ORDERED and LIST_MUTABLE) are also single-field Elements.

11.3 SOURCE REFERENCE SERVICE (//BLP/SRCREF)

The Source Reference and Tick Size Subscription services (aka `//blp/srcref`) are 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), which 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 NONBPS 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:

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

where <topic> is comprised of "<topic type>/<topic key>". The table below provides further details.

Listed below are the source reference string definitions:

Source Reference Name	Description
<service owner>	For B-PIPE is "blp"
<service name>	Source Reference and Tick Size Subscription service name is "/srcref"
<subservice name>	/brokercodes, /conditioncodes, /tradingstatuses or /ticksizes
<topic type>	/eid
<topic key>	EID-Number (FEED_EID1 => FEED_EID4)

Currently four subservices can be used in a user's Subscription string. Listed below are the 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 Subscriptions only. Here are the possible filters available for each:

Filter Name (type)	Subscription String Format
Subservice Name: /conditioncodes	

TRADE QUOTE
MKTDEPTH
TRADE,QUOTE TRADE,MKTDEPTH
QUOTE,MKTDEPTH
TRADE,QUOTE,MKTDEPTH
	//blp/srcref/conditioncodes/eid/<eid>?type=TRADE
	//blp/srcref/conditioncodes/eid/<eid>?type=QUOTE
	//blp/srcref/conditioncodes/eid/<eid>?type= MKTDEPTH
	//blp/srcref/conditioncodes/eid/<eid>?type=TRADE,QUOTE
	//blp/srcref/conditioncodes/eid/<eid>?type= TRADE,MKTDEPTH
	//blp/srcref/conditioncodes/eid/<eid>?type= QUOTE,MKTDEPTH
	//blp/srcref/conditioncodes/eid/<eid>?type= TRADE,QUOTE,MKTDEPTH

Subservice Name: /tradingstatuses	

PERIOD
STATUS
PERIOD,STATUS
	//blp/srcref/tradingstatuses/eid/<eid>?type=PERIOD
	//blp/srcref/tradingstatuses/eid/<eid>?type=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.

11.3.1 IMPORTANT BPOD UPGRADE NOTES

1. B-PIPE breaks down Subscriptions into a more granular format. With BPOD, users would have subscribed to "//blp/mktref/srcref/eid/<eid>" to obtain all source references for that EID, including the broker codes, trade condition codes, quote condition codes, market-depth condition codes, period suspense codes, security suspense codes and tick sizes. Using B-PIPE, users 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>"

   "//blp/srcref/ticksizes/eid/<eid>".

2. B-PIPE has introduced filters for some of its subservices to allow users to subscribe to the data they are most interested in.
3. With B-PIPE, a description Message is returned for each subservice's sources.
4. With B-PIPE, Bloomberg now offers intraday updating for tick size changes.
5. If users are looking for the sources on contributor EIDs (or any EID), they should subscribe to //blp/srcref for any of the subservices (e.g., /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 no tick sizes information will be included.

11.3.2 CODE EXAMPLE

A SourceRefSubscriptionExample is found in the B-PIPE SDK for C++, Java and .NET. This C++ example demonstrates how to make a simple Source Reference Subscription for the condition codes associated with EID 14003. Displayed is the C++ code snippet — subscribing for a list of condition codes for EID 14003.

```
const char *list = "//blp/srcref/conditioncodes/eid/14003";
SubscriptionList subscriptions;

subscriptions.add(list, CorrelationId((char *)security));
session.susbcribe (subscriptions);
```

11.3.3 RESPONSE OVERVIEW

The Source Reference response will be a series of SUBSCRIPTION_DATA Events that users will be familiar with if they have developed Bloomberg API applications using any of the other streaming services such as `//blp/mktdata`, `//blp/mktlist` or `//blp/mktdepthdata`.

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 users are subscribing to.

The table below lists the possible enumeration values for the Event type and Event sub-type fields:

Name	Description	Values
SOURCE_REF_EVENT_TYPE_RT	Specifies 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	Specifies Event sub-type	Possible enumeration values: INITPAINT — Initial Paint REFRESH — Daily Refresh[a] UPDATE — Intraday Update

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

The subservice name included in the user's Subscription dictates which Event type (SOURCE_REF_EVENT_TYPE_RT) field items will be returned as initial snapshot (INITPAINT) and refresh sub-type messages. The table below tells users which SOURCE_REF_EVENT_TYPE_RT field types to expect based on the subservice in their Subscription.

11.3.4 RESPONSE EVENT TYPES BY SUBSERVICE

The table below lists the entire initial snapshot and refresh (i.e., INITPAINT and REFRESH, respectively) Event type fields users should expect to receive for the subservice they subscribe to.

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 the subservice, please see the table below.

11.3.5 BREAKDOWN OF EVENT TYPE FIELDS

The table below 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 above] will have an associated field array name of DESCRIPTIONS).

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.
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).

Field Name	Type	Contents
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 of 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.

11.3.6 HANDLING MULTIPLE MESSAGES (A.K.A. FRAGMENTS)

- Initial Paint Messages can be split into one or more smaller Messages when the returned data is too large to fit into a single Message. Users are responsible for handling this in their application.
- Users will achieve the above 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 in 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 the last message of FRAGMENT_END. If the SourceReferenceUpdates is split into more than two parts, all middle Fragments will be of type FRAGMENT_INTERMEDIATE. Displayed below are the 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

11.3.7 DATA RESPONSE FOR SUBSCRIPTION

Below is the 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
              }
       }
}
---------------------------------------
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_TICk_SIZE_RT = 0.000000
                  TICK_SIZE_TBL_BAND_TICk_SIZE_RT = 10000000000.000000
              }
       }
}

************************************************************************
* 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
                    }
            }
    }
    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_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_TICK_SIZE_RT = 0.000000
                        TICK_SIZE_TBL_BAND_TICK_SIZE_RT = 10000000000.000000
                    }
            }
    }
    ***************************************************************************
    * 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
                    }
            }
    }
    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_TABLE_ROW = { TICK_SIZE_TABLE_PRICE_TYPE_RT =
                        ABSOLUTE
                    }
            }
}
```

```
**********************************************************************
*  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

                }

        }
}
```

In the above sample output, a Subscription containing the subservice "/ticksizes" was made, thus a user 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, users will also receive "UPDATE" Event type Messages, which are unique to the /ticksizes subservice. However, no UPDATE "DESCRIPTION" Message will be sent.

Taking a look at the sample output above, users 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 to that Event type. For instance, in the Message containing the Event type "TICK_SIZE_TABLE", they will find an array of "TICK_SIZE_TABLES" fields.

12 Authorization and Permissioning (//blp/apiauth)

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

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

12.1 AUTHORIZATION_STATUS, REQUEST_STATUS, RESPONSE AND PARTIAL_RESPONSE EVENTS

REQUEST: AUTHORIZATIONREQUEST

Scenario	Message Type	Category	Sub-Category
User authorized successfully.	AuthorizationSuccess		
User not logged in to Bloomberg.	AuthorizationFailure	NO_AUTH	NOT_LOGGED_IN
Invalid User ID	AuthorizationFailure	BAD_ARGS	INVALID_USER
Valid User ID belonging to different firm	ResponseError	NO_AUTH	CROSS_FIRM_AUTH
Invalid Display (when IP is specified).	AuthorizationFailure	NO_AUTH	INVALID_DISPLAY
Timeout waiting for input or expired token.	AuthorizationFailure	NO_AUTH	TOKEN_EXPIRED
Bad unparsable token supplied.	AuthorizationFailure	NO_AUTH	BAD_AUTH_TOKEN
User cancels request (Launchpad).	AuthorizationFailure	NO_AUTH	CANCELLED_BY_USER
UserAsidEquivalence check failed.	AuthorizationFailure	NO_AUTH	ENTITLEMENTS_MISMATCH
No token and IP specified.	ResponseError	BAD_ARGS	N/A
User has logged off and then back on to the Bloomberg Professional service. User's Identity object remains valid. message = "User re-logged on"	EntitlementChanged	N/A	N/A
Entitlements of the user/ application have been changed in EMRS. An hour usually needed to take effect and, therefore, to generate the Message. User/application's Identity object remains valid. Message = "Administrative Action"	EntitlementChanged	N/A	N/A
User logs in to a Bloomberg Professional service other than the one on the PC running his application.	AuthorizationRevoked	NO_AUTH	INVALID_DISPLAY

Scenario	Message Type	Category	Sub-Category
User uses an API that is either deprecated or passes parameters in an authorization request that are not supported for the specific product. For example, emrsname + IP authorization is not supported for ServerApi. Similarly, UUID+IP authorizations are not supported on for all products. A descriptive error Message is returned in the latter case.	AuthorizationFailure	NOT_AVAILABLE	NOT_AVAILABLE_API
User locked out of the Bloomberg Professional service.	AuthorizationRevoked	NO_AUTH	LOCKOUT
Sent when deactivating the application in EMRS after it had been used to authenticate in APPLICATION_ONLY mode. Also sent when unchecking the activate checkbox in EMRS for the user after it had been authenticated. Message = “Administrative Action”	AuthorizationRevoked	NO_AUTH	CANCELED_BY_SERVER
User logs in to a Bloomberg Professional service other than the one on the PC running his application.	AuthorizationRevoked	NO_AUTH	INVALID_DISPLAY

12.2 REQUEST_STATUS, RESPONSE AND PARTIAL_RESPONSE EVENTS

Message Type	Scenario	Category	Sub-Category
AuthorizationUpdate	User logged in to another Bloomberg Professional service.	NO_AUTH	INVALID_DISPLAY
AuthorizationUpdate	User locked out of Bloomberg Professional service. Click here for further details.	NO_AUTH	LOCKOUT
AuthorizationUpdate	Authorization cancelled by the server through EMRS administrator.	UNCLASSIFIED	CANCELLED_BY_SERVER
AuthorizationRequest	User not permitted to use the application.	NO_AUTH	NO_APP_PERM
AuthorizationRequest	Requested authorization type not supported for this ASID type.	NO_AUTH	INVALID_ASID_TYPE
AuthorizationRequest	User's authorization token has been used by another instance.	NO_AUTH	CREDENTIAL_REUSE
AuthorizationRequest	Token has expired. User must regenerate the token and authorize.	NO_AUTH	EXPIRED_AUTHTOKEN
AuthorizationRequest	Maximum number of devices for this seat type has been exceeded.	LIMIT	MAX_DEVICES_EXCEEDED
AuthorizationFailure	Exceeded maximum number of simultaneous authorizations.	LIMIT	n/a
AuthorizationUpdate	Entity/ASID delivery point not enabled in EMRS. This error Message received if a failure is dynamically detected because someone changed EMRS and an existing authorization is affected after the authorization had been successfully made.	NO_AUTH	EMRS_ENTITY_ASID_MISMATCH
AuthorizationFailure	Entity/ASID combination not enabled in EMRS. This error Message received if this failure is detected at authorization time.	NO_AUTH	EMRS_ENTITY_ASID_MISMATCH
AuthorizationFailure	Application IP mismatch with EMRS IP ranges.	NO_AUTH	EMRS_IPRANGE_MISMATCH
AuthorizationFailure	User or application not enabled for datafeed (B-PIPE) access in EMRS and attempting to authorize using B-PIPE.	NO_AUTH	EMRS_DATAFEED_DISABLED
AuthorizationFailure	User or application not enabled for DDM access in EMRS and attempting to authorize using a DDM server.	NO_AUTH	EMRS_PLATFORM_DISABLED
AuthorizationFailure	Application has no instance created for the B-PIPE instance (delivery point) in EMRS.	NO_AUTH	INVALID_DELIVERY_POINT
AuthorizationFailure	Application is authorizing from a machine whose IP is being prevented by the IP Restrictions configured in EMRS.	NO_AUTH	IP_NOT_IN_RANGE

12.3 TOKEN_STATUS EVENT

Message Type	Scenario	Category	Sub-Category
TokenGenerationSuccess	A token successfully generated.	N/A	N/A
TokenGenerationFailure	Library or backend errors	NO_AUTH	INTERNAL_ERROR
TokenGenerationFailure	User not found in EMRS database.	NO_AUTH	INVALID_USER
TokenGenerationFailure	Application name not found in EMRS database.	NO_AUTH	INVALID_APP
TokenGenerationFailure	Firm number mismatches with user(s) or application(s).	NO_AUTH	CROSS_FIRM_AUTH
TokenGenerationSuccess	A token successfully generated.		
TokenGenerationFailure	Token generation unsuccessful. generated.	BAD_ARGS	INVALID_USER or INVALID_APP