Enterprise Transport API Java Edition 3.6.0.L1

REFINITIV DOMAIN MODEL USAGE GUIDE

Document Version: 3.5.0.1
Date of issue: October 2020
Document ID: ETAJ360UMRDM.200

REFINITIV™

Contents

List of Figures

List of Tables

1 Introduction

1.1 About this Manual

This manual describes how the Refinitiv Domain Models are defined in terms of the Open Message Model. Data conforming to Refinitiv Domain Models are available via Refinitiv Real-Time Distribution System, Refinitiv Real-Time, and Refinitiv Data Feed Direct (RDF-D) using the Enterprise Transport API.

1.2 Audience

This guide is written for software developers who are familiar with the Enterprise Transport API and want to develop Enterprise Transport API-based applications to access Refinitiv Domain Model-formatted data. Before reading this manual:

- Users should be familiar with Open Message Model concepts and types.
- It may be useful to read the *Enterprise Transport API Java Edition Developers Guide* and be familiar with the example applications provided in the Enterprise Transport API package.

1.3 Open Message Model

The ***Open Message Model*** is a collection of message header and data constructs. Some Open Message Model message header constructs, such as the Update message, have implicit market logic associated with them while others, such as the Generic message, allow for free-flowing bi-directional messaging. Open Message Model data constructs can be combined in various ways to model data that ranges from simple (or flat) primitive types to complex multiple-level hierarchal data.

The layout and interpretation of any specific Open Message Model, also referred to as a domain model, is described within that model's definition and is not coupled with the API. The Open Message Model is the flexible tool that provides the building blocks to design and produce domain models to meet the needs of the system and its users. The Enterprise Transport API provides structural representations of Open Message Model constructs and manages the Refinitiv Wire Format binary-encoded representation of the Open Message Model. Enterprise Transport API users can leverage the provided Open Message Model constructs to consume or provide Open Message Model data throughout their Refinitiv Real-Time Distribution System.

1.4 Refinitiv Wire Format

Refinitiv Wire Format is the encoded representation of the Open Message Model. Refinitiv Wire Format is a highly-optimized, binary format designed to reduce the cost of data distribution as compared to previous wire formats. Binary encoding represents data in the machine's native manner, enabling further use in calculations or data manipulations. Refinitiv Wire Format allows for serializing Open Message Model message and data constructs in an efficient manner while still allowing rich content types. Refinitiv Wire Format can distribute field identifier-value pair data, self-describing data, as well as more complex, nested hierarchal content.

1.5 References

For additional Enterprise Transport API documentation, refer to:

- The *Enterprise Transport API Java Edition Developers Guide*
- The Refinitiv Developer Community

1.6 Documentation Feedback

While we make every effort to ensure the documentation is accurate and up-to-date, if you notice any errors, or would like to see more details on a particular topic, you have the following options:

- Send us your comments via email at apidocumentation@refinitiv.com.
- Mark up the PDF using the Comment feature in Adobe Reader. After adding your comments, you can submit the entire PDF to Refinitiv by clicking **Send File** in the **File** menu. Use the apidocumentation@refinitiv.com address.

1.7 Conventions

1.7.1 Typographic

The Enterprise Transport API uses the following typographical conventions:

- The Refinitiv Domain Models are described in terms of Open Message Model concepts. Images and XML example layouts are provided as a reference in relevant sections.
- In-line Classes, methods, and types are shown in **`orange, Lucida Console`** font.
- Parameters, filenames, and directories are shown in **Bold** font.
- Document titles and variable values are shown in *italics*.
- When included in the body of the text, new concepts are called out in ***Bold, Italics*** the first time they are mentioned.
- Verbose code examples (one or more lines of code) are shown in Lucida Console font against an orange background. Comments in such code samples are formatted in green coloring. For example:

```
/* decode contents into the filter list object */
if ((retVal = filterList.decode(decIter)) >= CodecReturnCodes.SUCCESS)
{
    /* create single filter entry and reuse while decoding each entry */
    FilterEntry filterEntry = CodecFactory.createFilterEntry();
```

1.7.2 General Transport API Syntax

The Enterprise Transport API uses the following general API syntax conventions:

- Dot-separated notation indicates data available within a hierarchy. Each period can indicate a Class (e.g., **RefreshMsg.msgKey**), a data member (e.g., **msgKey.name**), an entry (e.g., **RefreshMsg.Payload.DirectoryInfo**, where **DirectoryInfo** refers to the **DirectoryInfo** filter entry), or an element name (e.g., **UpdateMsg.Payload.DirectoryInfo.name** where **name** refers to the name element).
- **streamId** values are assigned by the application and used across all domain models. Consumer applications assign positive **streamId** values when requesting content and interactive provider applications respond using the same **streamId**. Non-interactive provider applications assign negative **streamId** values.
- ***Payload*** generically refers to the message payload. On any **Msg**, the payload is housed in the **encodedDataBody** buffer and follows the formatting rules of that message's **containerType**.
- Integer constants are defined in all capital letters with underscores (e.g., **DomainTypes.MARKET_PRICE**, **Directory.ServiceFilterIds.INFO**). In the Enterprise Transport API, they can be found in the **com.refinitiv.eta.rdm** package.
- The names of Enterprise Transport API **FilterEntry.id** values (e.g. **Directory.ServiceFilterIds.INFO**) correspond to the flag value enumeration defined for use with the message key's **filter** (e.g., **Directory.ServiceFilterFlags.INFO**). Names may be shortened for clarity (e.g., **ServiceInfo**).

1.7.3 Definitions and Standard Behaviors

This Enterprise Transport API manual uses the following terms and the API illustrates the following default behavior:

- ***Not Used*** means the attribute is not extensible; the Enterprise Transport API may pass-on the information, however there is no guarantee that the data will be passed through the network now or in the future. Use of a "Not Used" attribute may cause problems when interacting with some components.
- ***Required*** means the data must be provided or set.
- ***Conditional*** means date might be required depending on a particular scenario or context. Refer to the description for specific details.
- ***Optional*** means the data may be provided or set, but is not required. This data should be handled and understood by all applications, even if not including it. When present, this information should be passed through the network.
- If data is not present, the Enterprise Transport API assumes the default value.
- Generic message use is not supported within existing, defined Refinitiv Domain Models, except when explicitly defined.
- Posting is assumed to be supported within currently-defined Refinitiv Domain Models, except when otherwise indicated. Posting is not supported on Source Directory and Dictionary domains. Posting within the Login domain must follow off-stream posting rules and target a domain other than Login. Posting on any other allowed domains must follow on-stream posting rules and target that specific domain. For further details about posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

1.8 Acronyms and Abbreviations

ACRONYM	DEFINITION
AAAAPI	Authorization, Authentication, and Administration API
ADH	Refinitiv Real-Time Advanced Data Hub
ADS	Refinitiv Real-Time Advanced Distribution Server
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange

Table 1: Acronyms and Abbreviations

ACRONYM	DEFINITION
ATS	Refinitiv Real-Time Advanced Transformation Server
DACS	Data Access Control System
DMM	Domain Message Model
EMA	Enterprise Message API
ETA	Enterprise Transport API
OMM	Open Message Model
QoS	Quality of Service
RDF-D	Refinitiv Datafeed Direct
RDM	Refinitiv Domain Model
RDP	Refintiiv Data Platform
RMTES	Multi-lingual text encoding standard
RSSL	Refinitiv Source Sink Library
RTT	Round Trip Time; this definition is used for the round trip latency monitoring feature.
RWF	Refinitiv Wire Format

Table 1: Acronyms and Abbreviations (Continued)

1.9 What's New

This release of the Refinitiv Domain Model Usage Manual includes general documentation changes. Appendix B maintains a versioned list of changes to this document.

2 Domain Model Overview

2.1 What is a Domain Message Model?

A ***Domain Message Model*** describes a specific arrangement of Open Message Model message and data constructs. A domain message model will define any specialized behaviors associated with the domain or any specific meaning or semantics associated with data contained in the message. Unless a domain model specifies otherwise, any implicit market logic associated with a message still applies (e.g. an Update message indicates that any previously-received data also contained in the Update message is being modified).

2.2 Refinitiv Domain Models Vs User-Defined Models

2.2.1 Refinitiv Domain Models

A Refinitiv Domain Model is a domain message model typically provided or consumed by a Refinitiv product, such as the Refinitiv Real-Time Distribution System, Refinitiv Data Feed Direct, or the Refinitiv Data Platform. Some currently-defined Refinitiv Domain Models allow for authenticating to a provider (e.g. Login), exchanging field or enumeration dictionaries (e.g. Dictionary), and providing or consuming various types of market data (e.g. Market Price, Market by Order, Market by Price). Refinitiv's defined models have a domain value of less than 128.

The following table provides a high-level overview of the currently-available Refinitiv Domain Models. The following chapters provide more detailed descriptions for each of these.

DOMAIN	PURPOSE
Login	Authenticates users and advertise/request features that are not specific to a particular domain. Use of and support for this domain is required for all Open Message Model applications. This is considered an administrative domain, content is required and expected by many Refinitiv components and conformance to the domain model definition is expected. For further details refer to 3, Login Domain.
Source Directory	Advertises information about available services and their state, QoS, and capabilities. This domain also conveys any group status and group merge information. Interactive and non-Interactive Open Message Model provider applications require support for this domain. Refinitiv strongly recommends that Open Message Model consumers request this domain. This is considered an administrative domain, and many Refinitiv components expect and require content to conform to the domain model definition. For further details, refer to 4, Source Directory Domain.
Dictionary	Provides dictionaries that may be needed when decoding data. Though use of the Dictionary domain is optional, Refinitiv recommends that provider applications support the domain's use. Considered an administrative domain, content is required and expected by many Refinitiv components and following the domain model definition is expected. For further details refer to 5, Dictionary Domain.
Market Price	Provides access to Level I market information such as trades, indicative quotes and top of book quotes. Content includes information such as volume, bid, ask, net change, last price, high, and low. For further details refer to 6, Market Price Domain.
Market By Order	Provides access to Level II full order books. Contains a list of orders (keyed by the order IDs) with related information such as price, whether it is a bid/ask order, size, quote time, and market maker identifier. For further details refer to 7, Market By Order Domain.
Market By Price	Provides access to Level II market depth information. Contains a list of price points (keyed by that price and the bid/ask side) with related information. For further details refer to 8, Market By Price Domain.

Table 2: Refinitiv Domain Model Overview

DOMAIN	PURPOSE
Market Maker	Provides access to market maker quotes and trade information. Contains a list of market makers (keyed by that market maker's ID) with related information such as that market maker's bid and asking prices, quote time, and market source. For further details refer to 9, Market Maker Domain.
Yield Curve	Provides access to yield curve information. This can contain input information used to calculate a yield curve along with output information (which is the curve itself). A yield curve shows the relation between the interest rate and the term associated with the debt of a borrower. The curve's shape can help to give an idea of future economic activity and interest rates. For further details refer to 10, Yield Curve Domain.
Symbol List	Provides access to a set of symbol names, typically from an index, service, or cache. Minimally contains symbol names and can optionally contain additional cross-reference information such as permission information, name type, or other venue-specific content. For further details refer to 11, Symbol List Domain.

Table 2: Refinitiv Domain Model Overview (Continued)

2.2.2 User-Defined Domain Model

A ***User Defined Domain Model*** is a domain message model defined by a party other than Refinitiv. These may be defined to solve a specific user or system need in a particular deployment which is not resolvable through the use of a Refinitiv Domain Model. Any user-defined model must use a domain value between 128 and 255. If needed, domain model designers can work with Refinitiv to define their models as standard Refinitiv Domain Models. This allows for the most seamless interoperability with future Refinitiv Domain Model definitions and with other Refinitiv products.

2.2.3 Domain Message Model Creation

This document discusses Refinitiv Domain Models capable of flowing through the Enterprise Transport API. Enterprise Transport API users can leverage the Open Message Model to create their own Domain Message Models in addition to those described in this document. When defining a Domain Message Model, consider the following questions / points:

- Is a new Domain Message Model really needed, or can you express the data in terms of an existing Refinitiv Domain Model?
- The Domain Message Model should be well-defined. Following the design templates used in this document is a good approach. The structure, properties, use cases, and limitations of the Domain Message Model should be specified.
- While the Open Message Model provides building blocks that can structure data in many ways, the semantics of said data must abide by the rules of the Open Message Model. For example, custom Domain Message Models should follow the request, refresh, status, and update semantics implicitly defined by those messages. If more flexible messaging is desired within a custom Domain Message Model, it can be accomplished through the use of a generic message, which allows for more free-form bidirectional messaging after a stream is established.
- **`domainType`** values less than 128 are reserved for Refinitiv Domain Models. The **`domainType`** of a custom Domain Message Model must be between 128 and 255.
- You might want to work with Refinitiv to define a published Refinitiv Domain Model, rather than use a custom Domain Message Model. This ensures the most seamless interoperability with future Refinitiv Domain Models and other Refinitiv products.

2.3 Consumer / Interactive Provider Initial Interaction

An Open Message Model consumer application can connects to Open Message Model interactive provider applications, including the Refinitiv Real-Time Distribution System, Refinitiv Data Feed Direct, and the Refinitiv Data Platform. This interaction first requires an exchange of login messages between the consumer and provider, where the provider can either accept or reject the consumer. If the consumer is allowed to log in, it may then request the list of services available from the provider. Optionally[1], the consumer can request any dictionaries it needs to decode data from the provider. After this process successfully completes, the consumer application can begin requesting from non-administrative domains, which provide other content (e.g. Market Price, Market By Order).

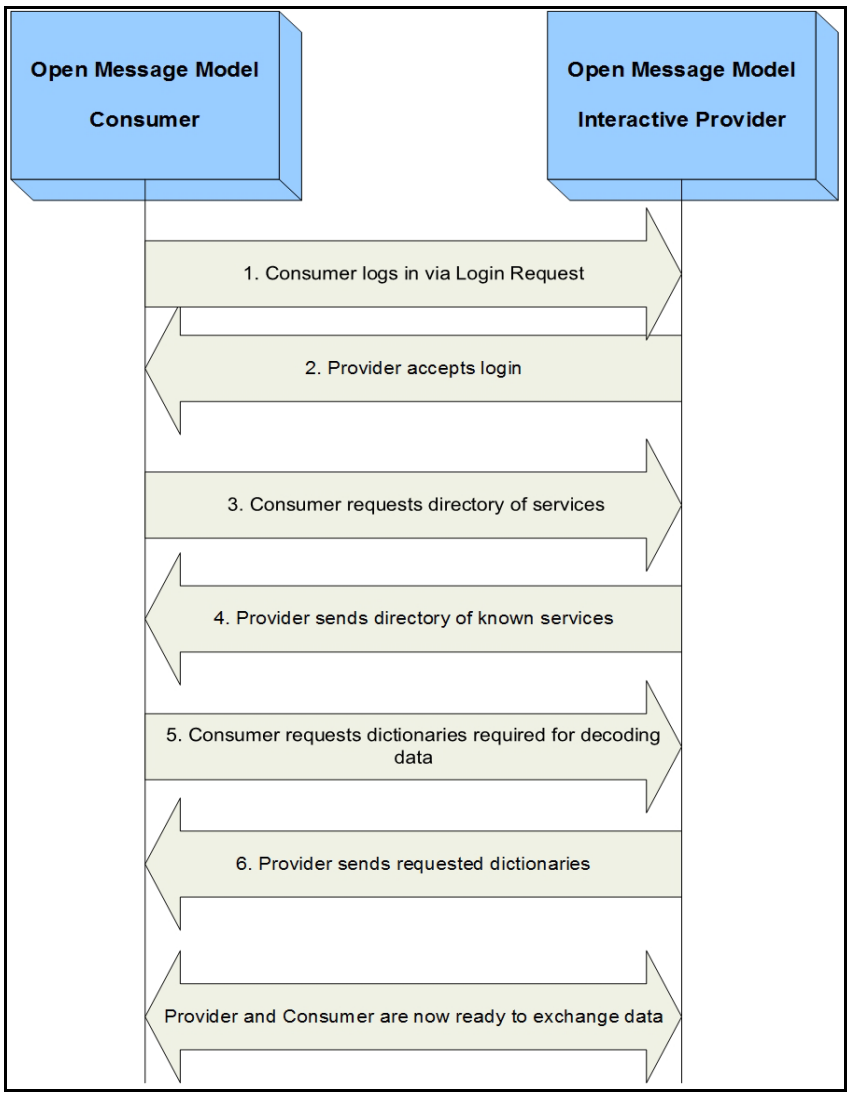


Figure 1. Open Message Model Consumer and Interactive Provider Initial Interactions

1. Instead of downloading any needed dictionaries, the application can load them from a local file.

2.4 Non-Interactive Provider Initial Interaction

An Open Message Model non-interactive provider application can establish a connection to a Refinitiv Real-Time Advanced Data Hub. After successfully connecting, an Open Message Model non-interactive provider can publish information into the Refinitiv Real-Time Advanced Data Hub's cache, without needing to handle requests for the information. This interaction first requires a login message exchange between the non-interactive provider and the Refinitiv Real-Time Advanced Data Hub, where the hub can either accept or reject the non-interactive provider. If the non-interactive provider logs in, it should push out its service information. After this process successfully completes, the non-interactive provider application can begin publishing content on other non-administrative domains (e.g. Market Price, Market By Order).

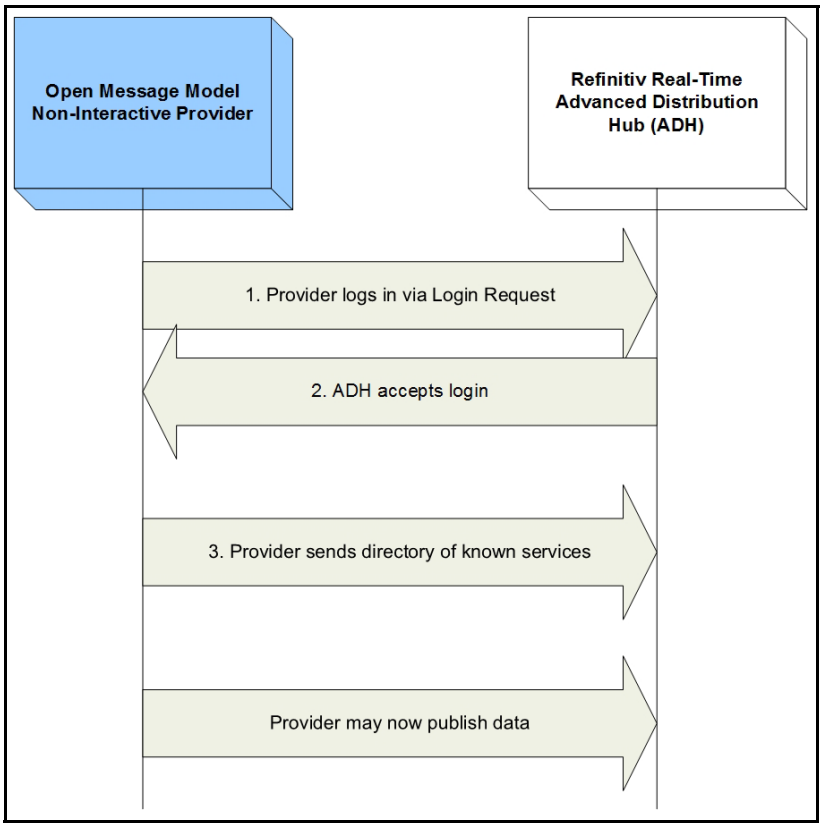


Figure 2. Open Message Model Non-Interactive Provider Initial Interaction

2.5 Sending and Receiving Content

Use of non-administrative domains generally follows a specific sequence:

- The consumer sends an **`RequestMsg`** containing the name of an item it is interested in.
- The provider first responds with an **`RefreshMsg`** to bring the consumer up to date with all currently available information.
- As data changes, the provider sends **`UpdateMsg`** (if the consumer requested streaming information).
- When the consumer is no longer interested, it sends an **`CloseMsg`** to close the stream (or, if the provider needs to close the stream, it uses an **`StatusMsg`**).

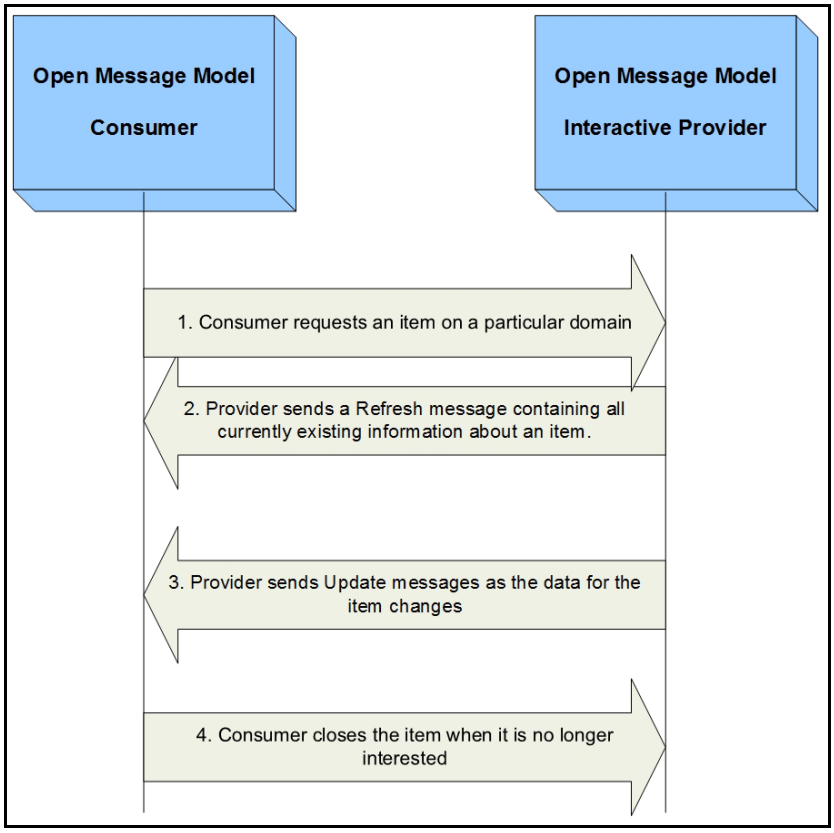


Figure 3. General Domain Use

2.6 General Enterprise Transport API Concepts

Many domains share a set of common behaviors for handling data. If a specific behavior is not supported on a domain, this should be specified in that domains detailed description. This section briefly defines these concepts; the *Enterprise Transport API Java Edition Developers Guide* describes them in greater detail.

2.6.1 Snapshot and Streaming Requests

Many domains generally support issuing a request message with or without setting the **RequestMsgFlags.STREAMING** flag. When the flag is set, the request is known as a "streaming" request, meaning that the refresh will be followed by updates.

When a snapshot request is made, the refresh should have a **`streamState`** of **StreamStates.NON_STREAMING**. When the final part of the refresh is received, the stream is considered closed (the final refresh is indicated by the **RefreshMsgFlags.REFRESH_COMPLETE** flag on the **`RefreshMsg`**). The consumer should be prepared to receive status messages or update messages between the first and final parts of the refresh (if the domain supplies only single part refresh messages, like Market Price, no updates would be delivered on the stream).

A provider may also set the **`streamState`** to **StreamStates.NON_STREAMING** if it does not allow streaming of the requested item.

2.6.2 Multi-Part Messages

Many domains support splitting the content of a large **`RefreshMsg`** into multiple, smaller refresh messages. Each part is a continuation of data from the previous parts, and the final part is denoted by including the **RefreshMsgFlags.REFRESH_COMPLETE** flag.

When possible, the provider should indicate the expected number of total container entries across all parts of the refresh. This is set using the **`totalCountHint`** parameter available on some containers (such as **`Map`**).

Though the Enterprise Transport API Transport Package can fragment large messages on the users behalf, Refinitiv recommends that the application fragment large messages whenever possible. The application knows best how data should be split, and this allows consuming applications to receive and process parts of the message without needing to wait for all parts to be delivered.

In addition to **`RefreshMsg`**, **`PostMsg`** and **`GenericMsg`** also have multi-part support. For more information, see the *Enterprise Transport API Java Edition Developers Guide*.

2.6.3 Reissue Requests and Pause/Resume

A consumer application can request a new refresh and change certain parameters on an already requested stream. To do so, the application sends a subsequent **`RequestMsg`** on the same stream. This is known as a ***reissue***.

A reissue changes the priority of a stream and pauses or resumes data flow.

- To pause streaming data, the application can send a reissue with the **RequestMsgFlags.PAUSE** flag. Issuing a pause on the Login stream is interpreted as a Pause All request, resulting in all streams being paused.
- To resume data flow on the stream, the application can send a subsequent reissue with the **RequestMsgFlags.STREAMING** flag. Issuing a resume on the Login stream is interpreted as a Resume All.

Pause and Resume is provided as a best effort, and data may continue streaming even after a pause has been issued.

For further details on reissue requests, changeable parameters, and Pause and Resume functionality, refer to the *Enterprise Transport API Java Edition Developers Guide*.

2.6.4 Rippling Fields

The **FieldList** container supports the ***rippling of fields***. When rippling, newly received content associated with a **fieldId** replaces previously received content associated with the same **fieldId**. The previously received content is moved to a new **fieldId**, which is typically indicated in a field dictionary. In the **RDMFieldDictionary**, the '**RIPPLES TO**' column defines **fieldId** information used when rippling (for details, refer to Section 5.5.1 and Section 5.5.2). Rippling typically reduces bandwidth consumption. Because previously received information is still relevant, it would normally need to be sent with subsequent updates even though the value does not change. The use of rippling allows this information to be optimized out of subsequent updates; however it relies on the consumer to use ripple information from a field dictionary to correctly propagate the previously received content.

NOTE: The consumer application is responsible for rippling and the Enterprise Transport API does not perform entry rippling.

WARNING! The application should not perform rippling if the **updateType** of the **UpdateMsg** is **UpdateEventTypes.CORRECTION**.

2.6.5 Dynamic View

A ***dynamic view*** allows a consumer application to specify a subset of data content in which it is interested. A providing application can choose to supply only this requested subset of content across all response messages. This filtering results in reduced data flow across the connection. View use can be leveraged across all non-administrative domain model types, where specific usage and support should be indicated in the model definition. The provider indicates its support for view requesting via the **supportViewRequests** Login attribute, as described in Section 3.3.1. For more information on dynamic views, refer to the *Enterprise Transport API Java Edition Developers Guide*.

2.6.6 Batch Request

A ***batch request*** allows a consumer application to indicate interest in multiple like-item streams with a single **RequestMsg**. A providing application should respond by providing a status on the batch request stream itself and with new individual item streams for each item requested in the batch. Batch requesting can be leveraged across all non-administrative domain model types. The provider indicates its support for batch requests via the **supportBatchRequests** Login attribute, as described in Section 3.3.1. For more information on batch requests, refer to the *Enterprise Transport API Java Edition Developers Guide*.

2.6.7 Posting

Posting offers an easy way for an Open Message Model consumer application to publish content to upstream components which can then provide the information. This can be done off-stream using the Login domain or on-stream using any other non-administrative domain. Use **PostMsg** to post content to the system. A **PostMsg** can contain any Open Message Model container type as its payload (but this is often a **Msg**). A provider indicates support for posting via the **supportOMMPost** Login attribute, as described in Section 3.3.1. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

3 Login Domain

3.1 Description

The ***Login*** domain registers a user with the system, after which the user can request[1], post[2], or provide[3] Refinitiv Domain Model content. A Login request can also be used to authenticate a user with the system.

- A consumer application must log into the system before it can request or post content.
- A non-interactive provider application must log into the system before providing any content. An interactive provider application is required to handle log in requests and provide Login response messages, possibly using the Data Access Control System to authenticate users.

For further details:

- Section 3.2 details the use of each message within the Login domain.
- Section 3.3 presents the message payloads.
- Section 3.4 includes a brief summary of login handling and authentication.
- Section 3.5 includes example layouts of login messages.

1. Consumer applications can request content after logging into the system.
2. Consumer applications can post content, which is similar to contribution or unmanaged publication, after logging into the system.
3. Non-interactive provider applications.

3.2 Usage

3.2.1 Login Request Message

A Login request message is encoded and sent by Open Message Model consumer and Open Message Model non-interactive provider applications. This message registers a user with the system. After receiving a successful login response, applications can then begin consuming or providing additional content. An Open Message Model provider can use the Login request information to authenticate users with the Data Access Control System.

An initial Login request must be streaming (i.e., a **RequestMsgFlags.STREAMING** flag is required). After the initial Login stream is established, subsequent Login requests using the same **`streamId`** can be sent to obtain additional refresh messages, pause the stream, or resume the stream. If a login stream is paused, this is interpreted as a 'Pause All' request which indicates that all item streams associated with the user should be paused. A login stream is paused by specifying **RequestMsgFlags.PAUSE** without the streaming flag. To resume data flow on all item streams (also known as a Resume All), the streaming flag must be sent again. For more information, refer to the *Enterprise Transport API Java Edition Developers Guide*.

For an example layout of this message, refer to Section 3.4.4.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.LOGIN == 1**
qos	Not used.
worstQos	Not used.
Priority.priorityClass	Not used.
Priority.priorityCount	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Optional, if present **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME_TYPE** should be specified. Possible values are: • **Login.UserIdTypes.NAME == 1** • **Login.UserIdTypes.EMAIL_ADDRESS == 2** • **Login.UserIdTypes.TOKEN == 3** • **Login.UserIdTypes.COOKIE == 4** • **Login.UserIdTypes.USER_AUTHN_TOKEN == 5** If **`msgKey.nameType`** is not set, it is assumed to be **Login.UserIdTypes.NAME**. A type of **Login.UserIdTypes.NAME** typically corresponds to a Data Access Control System user name. This can be used to authenticate and permission a user. **Login.UserIdTypes.TOKEN** is specified when using the AAA API The user token is retrieved from a AAA API gateway and then passed through the system via the **`msgKey.name`**. To validate users, a provider can pass this user token to an authentication manager application. If you specify **Login.UserIdTypes.USER_AUTHN_TOKEN**, you must also set **`msgKey.name`** to a single, null character (i.e., a 0x00 binary), and include an **`AuthenticationToken`** element in the **`msgKey.attrib`**. For details on the **`AuthenticationToken`**, refer to Section 3.2.2. Both **Login.UserIdTypes.TOKEN** and **Login.UserIdTypes.USER_AUTHN_TOKEN** can periodically change: when it changes, an application can send a login reissue to pass information upstream. • For further details on using **Login.UserIdTypes.TOKEN**, refer to the AAA API documentation. • For further details on using **Login.UserIdTypes.USER_AUTHN_TOKEN**, refer to the *UserAuthn Authentication User Manual*.[a]

Table 3: Login Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.name	**Required**. A **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME_TYPE** should be specified. **`msgKey.name`** should be populated with appropriate content corresponding to the **`msgKey.nameType`** specification.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Optional. If present, a **`msgKey.flags`** value of **MsgKeyFlags.HAS_ATTRIB** should be specified and **`msgKey.attribContainerType`** should be set to **DataTypes.ELEMENT_LIST**. Attributes are specified in Section 3.2.2.
Payload	Not used.

Table 3: Login Request Message Member Use (Continued)

a. For further details on Refinitiv Data Platform authentication, refer to the *UserAuthn Authentication User Manual*, accessible on MyRefinitiv in the Data Access Control System product documentation set.

3.2.2 Login Request Elements

You can use the Login **msgKey.attrib** elements to send additional authentication information and user preferences between components. The **ReqMsg.Attrib attribContainerType** is an **ElementList** (**DataTypes.ELEMENT_LIST**). The predefined elements available on a Login Request are shown in the following table.

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
AllowSuspectData	DataTypes.UINT	0 | 1	• **1**: Indicates that the consumer application allows for suspect **streamState** information • **0**: Indicates that the consumer application prefers any suspect data result in the stream being closed with an **StreamStates.CLOSED_RECOVER** state. For more information, refer to Section 3.4.4.
ApplicationAuthorizationToken	DataTypes.ASCII_STRING	Encrypted application authorization token	Indicates that application behaviors was inspected and approved by Refinitiv. Encrypt this token using the provided methods in the API to prevent other users or applications from compromising it. For more information on obtaining an application authorization token, contact your Refinitiv representative.
ApplicationId	DataTypes.ASCII_STRING	1 - 65535 e.g., **256**	The Data Access Control System application ID. If the server authenticates with the Data Access Control System, the consumer application might need to pass in a valid **ApplicationId**. This must be unique for each application. IDs from 1 to 256 are reserved for permanent market data applications. These are assigned by Refinitiv and will be uniform across all client systems. IDs from 257 to 65535 are available for site-specific use.
ApplicationName	DataTypes.ASCII_STRING	Name of application e.g., **Enterprise Transport API**	Identifies the application sending the Login request or response message. When present, the application name in the Login request identifies the consumer, and the application name in the Login response identifies the Open Message Model provider.
AuthenticationExtended	DataTypes.BUFFER	Any binary buffer	This is a binary buffer whose content will be passed to the token authenticator as an additional means for verifying a user's identity.

Table 4: Login Request msgKey.attrib Elements

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
AuthenticationToken	DataTypes.ASCII_STRING	Any ASCII String, e.g., **HOLDER**	**Conditional**. **`AuthenticationToken`** is a client-generated token that identifies the user when operating in an environment that uses UserAuthn Authentication. On login reissue messages, this field contains a new token intended to replace the previous one about to expire. If your Refinitiv Real-Time Distribution System has UserAuthn Authentication enabled, an **`AuthenticationToken`** is included in the message. For further details on UserAuthn Authentication, refer to the *UserAuthn Authentication User Manual*, accessible on MyRefinitiv in the Data Access Control System product documentation set. The default setting is: "" (an empty string).
DisableDataConversion	N/A	N/A	Reserved by Refinitiv.
DownloadConnectionConfig	DataTypes.UINT	0 | 1	Specifies whether to download the configuration: • **1**: Indicates the user wants to download connection configuration information. • **0** (or if absent): Indicates that no connection configuration information is desired. **0** is the default setting.
InstanceId	DataTypes.ASCII_STRING	Any ASCII String, e.g., **Instance1**	**`InstanceId`** is used to differentiate applications that run on the same machine. However, because **`InstanceId`** is set by the user logging into the system, it does not guarantee uniqueness across different applications on the same machine.
Password	DataTypes.ASCII_STRING	my_password	Sets the password for logging into the system. See specific component documentation to determine password requirements and how to obtain one.
Position	DataTypes.ASCII_STRING	ip addr/net e.g., **192.168.1.1/ net**	The Data Access Control System position. If the server is authenticating with the Data Access Control System, the consumer application might need to pass in a valid position.
ProvidePermissionExpressions	DataTypes.UINT	0 | 1	If specified on the Login Request, this indicates a consumer wants permission expression information to be sent with responses. Permission expressions allow for items to be proxy permissioned by a consumer via content-based entitlements. **`ProvidePermissionExpressions`** defaults to **1**.
ProvidePermissionProfile	DataTypes.UINT	0 | 1	When specified on a Login Request, indicates that a consumer desires the permission profile. An application can use the permission profile to perform proxy permissioning. **`ProvidePermissionProfile`** defaults to **1**.

Table 4: Login Request `msgKey.attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
Role	DataTypes.UINT	Login.RoleTypes.CONS == 0, Login.RoleTypes.PROV == 1	Indicates the role of the application logging onto the system. • An Open Message Model consumer application should specify its role as **Login.RoleTypes.CONS**. • An Open Message Model non-interactive provider application should specify its role as **Login.RoleTypes.PROV**. Open Message Model consumer applications typically connect to a different port number than non-interactive provider applications. Role information allows Refinitiv Real-Time Distribution System to detect and inform users of incorrect port use. **`Role`** defaults to **0**.
RoundTripLatency	DataTypes.UINT	2	Indicates whether the consumer supports Round Trip Time (RTT) latency monitoring. The presence of this element indicates that the consumer supports the RTT monitoring feature. Non-interactive providers do not use this element. If the element is missing, the consumer does not support RTT Latency monitoring.
SingleOpen	DataTypes.UINT	0 | 1	• **1**: Indicates the consumer application wants the provider to drive stream recovery. • **0**: Indicates that the consumer application will drive stream recovery. For more information, refer to Section 3.4.4. **`SingleOpen`** defaults to **1**.
SupportProviderDictionaryDownload	DataTypes.UINT	0 | 1	Indicates whether the server supports the Provider Dictionary Download feature: • **1**: The server supports provider dictionary downloads. • **0**: The server does not support provider dictionary downloads. If this element is missing, the server does not support provider dictionary downloads. For more information on the provider dictionary download feature, refer to the *Enterprise Transport API Java Edition Developers Guide*. **`SupportProviderDictionaryDownload`** defaults to **0**.

Table 4: Login Request `msgKey.attrib` Elements (Continued)

3.2.3 Login Refresh Message

A Login refresh message is encoded and sent by Open Message Model interactive provider applications. This message is used to respond to a Login Request message after the user's Login is accepted. An Open Message Model provider can use the Login request information to authenticate users with the Data Access Control System. After authentication, a refresh message is sent to convey that the login was accepted. If the login is rejected, a Login status message should be sent as described in Section 3.2.5.

The content of a Login Refresh message is expected to be atomic and contained in a single part, therefore **RefreshMsgFlags.REFRESH_COMPLETE** should be set to **true**. If the login refresh is sent in response to a request, the **RefreshMsgFlags.SOLICITED** flag should be set to **true** to indicate that this is a solicited response.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.LOGIN == 1**
state	When accepting Login: • **`streamState`** = StreamStates.OPEN • **`dataState`** = DataStates.OK • **`stateCode`** = StateCodes.NONE
qos	Not used.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Optional. If present, a **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME_TYPE** should be specified. Possible values: • **Login.UserIdTypes.NAME == 1** • **Login.UserIdTypes.EMAIL_ADDRESS == 2** • **Login.UserIdTypes.TOKEN == 3** • **Login.UserIdTypes.COOKIE == 4** • **Login.UserIdTypes.USER_AUTHN_TOKEN == 5** If **`nameType`** is not set then it is assumed to be a **`nameType`** of **Login.UserIdTypes.NAME**. If present, the value should match the type specified in the Login request.
msgKey.name	Optional. A **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified, and **`name`** should match the **`name`** specified in the Login request and contain appropriate content corresponding to the **`nameType`** specification.
msgKey.filter	Not used.
msgKey.identifier	Not used.

Table 5: Login Refresh Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.attrib	Optional. If present **`msgKey.flags`** value of **MsgKeyFlags.HAS_ATTRIB** should be specified and **`msgKey.attribContainerType`** should be **DataTypes.ELEMENT_LIST**. Elements are specified in Section 3.2.4.
Payload	Optional. Typically present when login requests connection configuration or permission profile information. The payload is sent as an **`ElementList`**. For payload details, refer to Section 3.3.1.

Table 5: Login Refresh Message Member Use (Continued)

3.2.4 Login Refresh Elements

The Login **msgKey.attrib** can be used to send additional authentication information and user preferences between components. The **attribContainerType** is an **ElementList**, which can contain any of the following predefined elements (none of which are required):

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
AllowSuspectData	DataTypes.UINT	0 | 1	Sets whether the provider application passes along suspect **streamState** information. • **1**: The provider application passes along suspect **streamState** information. **1** is the default setting. • **0**: The provider application does not pass along suspect data. Any suspect stream will be closed with an **StreamStates.CLOSED_RECOVER** state. For more information, refer to Section 3.4.4.
ApplicationId	DataTypes.ASCII_STRING	1 - 65535 e.g., **256**	Specifies the Data Access Control System application ID. If the server authenticates with the Data Access Control System, the consumer application may be required to pass in a valid **ApplicationId**. This should match whatever was sent in the request. This must be unique for each application. IDs from 1 to 256 are reserved for permanent market data applications. Refinitiv assigns these and they are uniform across all client systems. IDs from 257 to 65535 are available for site-specific use.
ApplicationName	DataTypes.ASCII_STRING	name of application e.g., **Enterprise Transport API** If connecting to a Refinitiv Real-Time Advanced Distribution Server, use **ADS**	Identifies the application sending the Login request or response message. When present, the application name in the Login request identifies the Open Message Model consumer and the application name in the Login response identifies the Open Message Model provider.
AuthenticationErrorCode	DataTypes.UINT	From **0** to **4294967296**	Specifies the code for a specific Refinitiv Real-Time Distribution System Authentication error (or non-error) condition. **0** indicates no error condition and is the default setting.
AuthenticationErrorText	DataTypes.ASCII_STRING	User-defined value	Text accompanying and explaining the **AuthenticationErrorCode**.
AuthenticationExtendedResp	DataTypes.BUFFER	User-defined value	This is a binary buffer. **AuthenticationExtendedResp** contains additional customer-defined data associated with the **AuthenticationToken** element sent in the original Login Request.
AuthenticationTTReissue	DataTypes.UINT	User-defined value	Indicates when a new authentication token needs to be reissued (in UNIX epoch time).

Table 6: Login Refresh msgKey.attrib Elements

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
Position	DataTypes.ASCII_STRING	ip addr/hostname or ip addr/net e.g.: **192.168.1.1/net**	Specifies the Data Access Control System location. If the server authenticates with the Data Access Control System, the consumer application might be required to pass in a valid position. If present, this should match whatever was sent in the request or be set to the IP address of the connected client.
ProvidePermissionExpressions	DataTypes.UINT	0 | 1	If specified on a Login Refresh, indicates that a provider will send permission expression information with its responses. **`ProvidePermissionExpressions`** is typically present because the login request message requested this information. Permission expressions allow for items to be proxy permissioned by a consumer via content-based entitlements. **`ProvidePermissionExpressions`** defaults to **1**.
ProvidePermissionProfile	DataTypes.UINT	0 | 1	If specified on the Login Refresh, indicates that the permission profile is provided. This is typically present because the login request message requested this information. An application can use the permission profile to perform proxy permissioning. **`ProvidePermissionProfile`** defaults to **1**.
RoundTripLatency	DataTypes.UINT	2	Indicates support for **`RoundTripLatency`** monitoring by the provider. If the element is missing, the provider might still support the feature.
SequenceRetryInterval	DataTypes.UINT	0-4,294,967,295	The Refinitiv Real-Time Advanced Distribution Server uses this element to configure a watchlist-enabled Enterprise Transport API reactor that consumes multicast data. Configures the number of seconds the reactor delays the recovery of items in response to a detected gap. **`SequenceRetryInterval`** defaults to **5**.
SequenceNumberRecovery	DataTypes.UINT	0 | 1	The Refinitiv Real-Time Advanced Distribution Server uses this element to configure a watchlist-enabled Enterprise Transport API Reactor that consumes multicast data. Configures whether the reactor recovers item streams when gaps are detected. • **0**: The reactor does not recover items. • **1**: The reactor recovers items. **1** is the default setting.

Table 6: Login Refresh `msgKey.attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
SingleOpen	DataTypes.UINT	0 | 1	Specifies whether the provider drives stream recovery: • **1**: The provider drives stream recovery. **1** is the default setting. • **0**: The provider does not drive stream recovery; it is the responsibility of the downstream application. For more information, refer to Section 3.4.4.
SupportBatchRequests	DataTypes.UINT	0, 7	Indicates whether the provider supports batch messages. Consumers use batch messages to specify multiple items or streams in the same request or close message. For more information on batch requesting, refer to the *Enterprise Transport API Java Edition Developers Guide*. • **0x0** (or if absent): The provider does not support batch messages. **0** is the default setting. • **0x1**: The provider supports batch request. • **0x2**: The provider supports batch reissue. • **0x4**: The provider supports batch close. For instance, if value is set to 7, then based on combination of bits set (0x1 + 0x2 + 0x4), provider supports batch request, reissue, and close.
SupportEnhancedSymbolList	DataTypes.UINT	0 | 1	Indicates whether the provider supports enhanced symbol list functionality. • **0**: The provider does not support Symbol List enhancements. **0** is the default setting. • **1**: The provider supports Symbol List data streams. For more information on Symbol List items, refer to the *Enterprise Transport API Java Edition Developers Guide*.
SupportOMMPost	DataTypes.UINT	0 | 1	Indicates whether the provider supports Open Message Model posting and whether the user is permissioned to post: • **1**: The provider supports Open Message Model posting and the user is permissioned. • **0**: The provider supports the Open Message Model posting feature, but the user is not permissioned. **0** is the default setting. • If absent, the server does not support the Open Message Model Post feature. For more information on Posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

Table 6: Login Refresh `msgKey.attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
SupportOptimizedPauseResume	DataTypes.UINT	0 | 1	Indicates whether the provider supports Optimized Pause and Resume. Optimized Pause and Resume allows for pausing/resuming of individual item streams or pausing all item streams (by pausing the Login stream). For more information on Pause and Resume, refer to the *Enterprise Transport API Java Edition Developers Guide*. • **1**: The server supports optimized pause and resume. • **0** (or if absent): The server does not support optimized pause and resume. **0** is the default setting.
SupportProviderDictionaryDownload	DataTypes.UINT	0 | 1	Indicates whether the server supports the Provider Dictionary Download feature: • **1**: The server supports the provider dictionary download. • **0**: The server does not support the provider dictionary download feature. **0** is the default setting. If this element is missing, the server does not support the provider dictionary download feature. For more information on the provider dictionary download feature, refer to the *Enterprise Transport API Java Edition Developers Guide*.
SupportStandby	DataTypes.UINT	0 | 1	Indicates whether the provider supports Warm Standby functionality. If supported, a provider can be told to run as an active or a standby server, where the active will behave as usual. The standby will respond to item requests only with the message header and will forward any state changing information. If informed that the active server failed, the standby begins sending responses and assumes active functionality. • **1**: The provider supports a Warm Standby group setup. • **0** (or if absent): The provider does not support warm standby functionality. **0** is the default setting. For more information on Warm Standby functionality, refer to Section 3.2.9.

Table 6: Login Refresh `msgKey.attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
SupportViewRequests	DataTypes.UINT	0 | 1	Indicates whether the provider supports requesting with Dynamic View information. Using Dynamic Views, a user can request only the specific contents of the response information in which they are interested. For more information on using Dynamic Views, refer to the *Enterprise Transport API Java Edition Developers Guide*. • **1**: The provider supports Dynamic Views specified on request messages. • **0** (or if absent): The provider does not support Dynamic Views specified on request messages. **0** is the default setting.
UpdateBufferLimit	DataTypes.UINT	0-4,294,967,295	The Refinitiv Real-Time Advanced Distribution Server uses this element to configure a watchlist-enabled Enterprise Transport API reactor that consumes multicast data. Configures the maximum number of multicast messages the reactor will interally buffer for an item. The reactor synchronizes buffered messages against the item's refresh. **`UpdateBufferLimit`** defaults to **100**.

Table 6: Login Refresh `msgKey.attrib` Elements (Continued)

3.2.5 Login Status Message

Open Message Model provider and non-interactive provider applications use the Login status message to convey state information associated with the login stream. Such state information can indicate that a login stream cannot be established or to inform a consumer of a state change associated with an open login stream.

The Login status message can also be used to reject a login request or close an existing login stream. When a login stream is closed via a status, any other open streams associated with the user are also closed as a result.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.LOGIN == 1**
state	When rejecting Login: • **`streamState`** = **StreamStates.CLOSED** or **StreamStates.CLOSED_RECOVER** • **`dataState`** = **DataStates.SUSPECT** • **`stateCode`** = **StateCodes.NOT_ENTITLED**
groupId	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.

Table 7: Login Status Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.nameType	Optional. If present, a **msgKey.flags** value of **MsgKeyFlags.HAS_NAME_TYPE** should be specified. Possible values: • **Login.UserIdTypes.NAME == 1** • **Login.UserIdTypes.EMAIL_ADDRESS == 2** • **Login.UserIdTypes.TOKEN == 3** • **Login.UserIdTypes.COOKIE == 4** If present, **msgKey.nameType** should match the type specified in the Login request. If **msgKey.nameType** is unspecified, it is assumed to be a **msgKey.nameType** of **Login.UserIdTypes.NAME**.
msgKey.name	Optional. A **msgKey.flags** value of **MsgKeyFlags.HAS_NAME** should be specified and **msgKey.name** should match the one used in the Login request and should contain appropriate content corresponding to the **msgKey.nameType** specification.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Optional. If present, contains the **AuthenticationErrorCode** and any associated **AuthenticationErrorText**.
Payload	Not used.

Table 7: Login Status Message Member Use (Continued)

3.2.6 Login Status Elements

The Login **msgKey.attrib** can be used to send additional authentication information and user preferences between components. The **attribContainerType** is an **ElementList**, which can contain any of the following predefined elements (none of which are required):

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
AuthenticationErrorCode	DataTypes.UINT	From **0** to **4294967296**	Specifies the code for a specific Refinitiv Real-Time Distribution System Authentication error (or non-error) condition. **0** indicates no error condition and is the default setting.
AuthenticationErrorText	DataTypes.ASCII_STRING		Text accompanying and explaining the **AuthenticationErrorCode**.
UpdateBufferLimit	DataTypes.UINT	0-4,294,967,295	The Refinitiv Real-Time Advanced Distribution Server uses this element to configure a watchlist-enabled Enterprise Transport API reactor that consumes multicast data. Configures the maximum number of multicast messages the reactor will interally buffer for an item. The reactor synchronizes buffered messages against the item's refresh. **UpdateBufferLimit** defaults to **100**.

Table 8: Login Status msgKey.attrib Elements

3.2.7 Login Update Message

Update messages are currently not used or supported on a Login stream.

3.2.8 Login Close Message

A Login close message is encoded and sent by Open Message Model consumer applications. This message allows a consumer to log out of the system. Closing a login stream is equivalent to a 'Close All' type of message, where all open streams are closed (thus all other streams associated with the user are closed). A provider can log off a user and close all of that user's streams via a Login Status message (for details, refer to Section 3.2.5).

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.CLOSE == 5**
domainType	**Required**. **DomainTypes.LOGIN == 1**
extendedHeader	Not used.
Payload	Not used.

Table 9: Login Close Message Member Use

3.2.9 Login Generic Message Use

3.2.9.1 Login Consumer Connection Status Message

A Login Consumer Connection Status message is encoded and sent by Open Message Model consumer applications. This message informs an interactive provider of its role in a Warm Standby group (either as an Active or Standby provider). When Active, a provider behaves normally. However, if a provider is Standby, it responds to requests only with a message header (intended to allow a consumer application to confirm the availability of their requested data across active and standby servers), and forwards any state-related messages (i.e., unsolicited refresh messages, status messages). While in Standby mode, a provider should aggregate changes to item streams whenever possible. If the provider is changed from Standby to Active via this message, all aggregated update messages are passed along. When aggregation is not possible, a full, unsolicited refresh message is passed along.

The consumer application is responsible for ensuring that items are available and equivalent across all providers in a warm standby group. This includes managing state and availability differences as well as item group differences.

Content for a Login Consumer Connection Status message is expected to be atomic and contained in a single part, therefore **GenericMsgFlags.MESSAGE_COMPLETE** should be present in the Generic message's flags.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.GENERIC == 7**
domainType	**Required**. **DomainTypes.LOGIN == 1**
partNum	Not used.
seqNum	Not used.
secondarySeqNum	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Not used.
msgKey.name	**Required**. **`msgKey.name`** must be set to **ConsumerConnectionStatus**.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. Indicates whether a provider acts as an Active or Standby server. Payload is sent as a **DataTypes.MAP** type. For further details, refer to Section 3.3.2.

Table 10: Login Consumer Connection Status Message Member Use

3.2.9.2 RTT Login Generic Message

A Round Trip Time (RTT) Login Generic Message exchange is initiated by the Interactive Provider application. This message must contain the **Ticks** count, which is set by the provider before sending the message to a consumer that supports RTT functionality. The CPU tick count can be retrieved using the **`{System.nanoTime()}`** call. When the consumer receives the RTT message, the consumer automatically sends it back to the interactive provider with the **Ticks** value unchanged. The interactive provider calculates the round trip time by subtracting the **Ticks** value from the message from its current time given by the **`{System.nanoTime()}`** call. In its subsequent RTT requests to the consumer, a provider can include the previously calculated **`RoundTripLatency`** value, in microseconds.

Handling RTT Login Generic messages on the provider's side should be implemented in the user application. On the consumer side, the Watchlist automatically mirrors the provider's RTT request back to the provider when RTT handling is configured. The consumer can listen for these messages and implement specific business-logic to further handle them.

COMPONENT	DESCRIPTION / VALUE
domainType	**Required**. **DomainTypes.LOGIN == 1**
Indications	**Required**. **`ProviderDriven`**: **true**, indicates that the message exchange was initiated by the provider and is not a response to a specific, consumer request. **`ProviderDriven`** should be set on both provider and consumer messages.
partNum	Not used.
seqNum	Not used.
secondarySeqNum	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Not used.
msgKey.name	Not used.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. **Payload** is sent as an **`ElementList`** type and must contain an **`ElementEntry`** with the ticks count that represents the time on the provider's side. Additionally, **Payload** can contain two optional entries: **`RoundTripLatency`** and **`TcpRetrans`**. For further details, refer to Section 3.3.2.

Table 11: RTT Login Generic Message Member Use

3.2.10 Login Post Message

Open Message Model consumer applications can encode and send data for any item via Post messages on the item's Login stream. This is known as ***off-stream posting*** because items are posted without using that item's dedicated stream. Posting an item on its own dedicated stream is referred to as ***on-stream posting***.

When an application is posting off-stream, the **`PostMsg`** requires **`msgKey`** information. For more details on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

3.2.11 Login Ack Message

Open Message Model provider applications encode and send acknowledgment messages (**`AckMsg`**) to acknowledge the receipt of Post messages. This message is used whenever a consumer posts and asks for acknowledgments. For more details on posting, see the *Enterprise Transport API Java Edition Developers Guide*.

3.3 Data

3.3.1 Login Refresh Message Payload

When a Login request message asks for connection configuration information (i.e., **DownloadConnectionConfig = 1**), a provider capable of supplying these details should respond with extended connection information in the **`RefreshMsg`** payload. This information can be useful for load balancing connections across multiple providers or Refinitiv Real-Time Advanced Distribution Server components. Load balancing can be set up in a manner where some well-known providers act solely as load-balancing servers, monitoring the load and state of other providers and directing consumers to less-loaded providers to handle the information exchange.

The extended connection information contains a list of other providers, along with connection and load-related information, and is formatted as a sorted **`Vector`** type, where each **`VectorEntry`** contains an **`ElementList`**. Each vector entry contains data specific to one provider. The summary data (an **`ElementList`**) contains information about the number of standby providers to which the consumer should connect. If this value is non-zero, the consumer is expected to support Warm Standby functionality and connect to multiple providers.

The list should be sorted in order of best to worst choice.

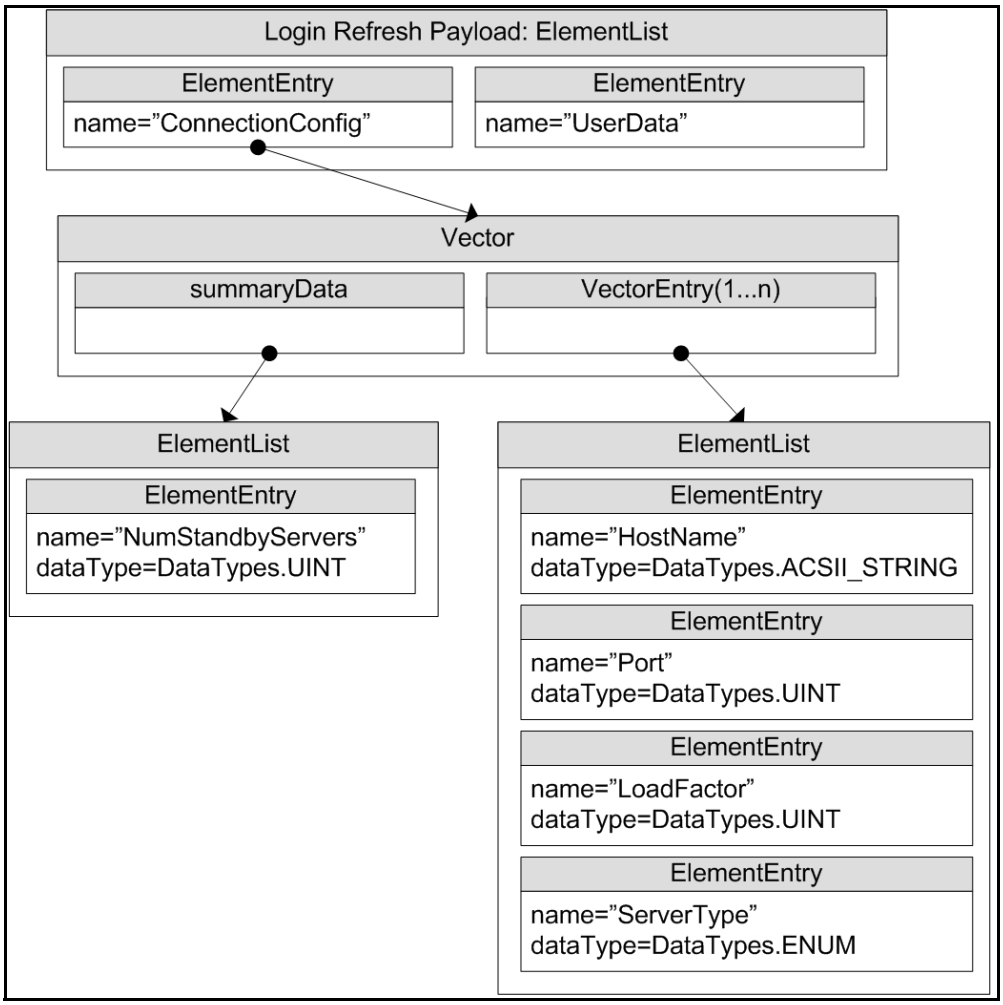


Figure 4. Login Refresh Message Payload

When the payload is present, the summary data **ElementList** must contain the following element (which has no default):

NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
NumStandbyServers	DataTypes.UINT	0 - 0xFFFFFFFF value	Specifies the number of standby servers to which the client can connect. If set to **0**, only one provider is connected, which serves as the primary connection (i.e., warm standby should not be attempted).

Table 12: Login Refresh.Payload Vector.SummaryData's ElementList Contents

Each **VectorEntry** contains an **ElementList**, each list describing a single provider. Possible elements in this list are as follows, with any default behavior included in the description:

NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Hostname	DataTypes.ASCII_STRING	"myHostName" "192.168.1.100"	**Conditional**. Specifies the candidate provider's IP address or hostname. **Hostname** is required when a payload is present.
Port	DataTypes.UINT	14002	**Conditional**. Specifies the candidate provider's port number. **Port** is required when a payload is present.
LoadFactor	DataTypes.UINT	0 - 65535	Describes the load of the provider, where **0** is the least loaded and **65535** is the most loaded. The **Vector** is expected to be sorted, so a consumer need not traverse the list to find the least loaded; the first **VectorEntry** should contain an **ElementList** describing the least-loaded provider. **LoadFactor** defaults to 65535.
ServerType	DataTypes.UINT	0 | 1	When using a warm standby setup, **ServerType** specifies the provider's expected behavior: • **0**: This provider should be the Active server. • **1**: This provider should be the Standby server. **1** is the default setting.
SystemID	DataTypes.ASCII_STRING		For future use.

Table 13: Login Refresh.Payload VectorEntry's ElementList Contents

3.3.2 Login Generic Message Payloads

3.3.2.1 Login Consumer Connection Status Message Payload

The Login data structure for **GenericMsg.Payload** is a Map of AsciiString -> ElementList. Each key is a **ServiceName.Map.Key**. Each **ElementList** contains one **ElementEntry**. The Login **GenericMsg.Payload** is formatted as an **Map** type, with a **keyPrimitiveType** of **DataTypes.ASCII_STRING** and a **containerType** of **DataTypes.ELEMENT_LIST**. There is no summary data and typically only one map entry that informs the provider of its warm standby role.

KEY PRIMITIVE TYPE	CONTAINER TYPE	PERMISSION DATA	DESCRIPTION
DataTypes.ASCII_STRING	DataTypes.ELEMENT_LIST	Not used	**Required**. This entry key must be set to **WarmStandbyInfo**.

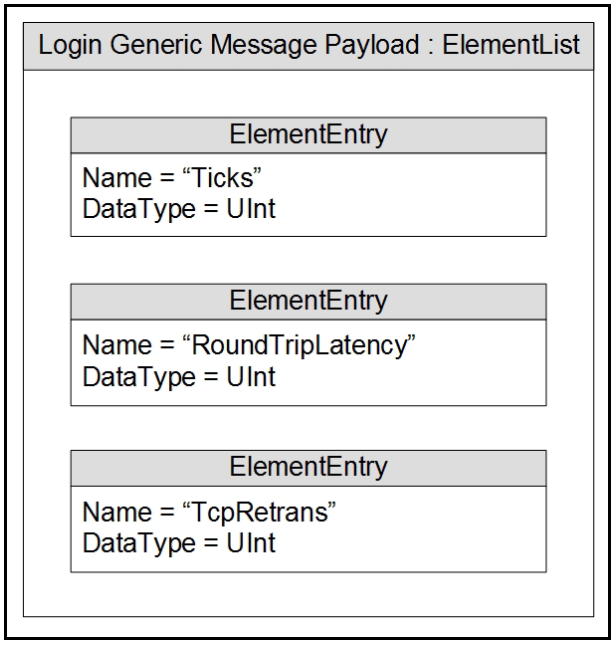
Table 14: GenericMsg.Payload Map's MapEntry Information

ELEMENT NAME	DATA TYPE	RANGE/EXAMPLE	DESCRIPTION
WarmStandbyMode	DataTypes.UINT	0 | 1	**Required**. Informs an interactive provider of its role in a Warm Standby group. • **0**: Informs the provider to be an Active server • **1**: Informs the provider to be a Standby server. **WarmStandbyMode** does not have a default.

Table 15: Login GenericMsg.Payload MapEntry Elements

3.3.2.2 RTT Login Generic Message Payload

The RTT message payload is an **`ElementList`**, which must contain an **`ElementEntry`** with **`Ticks`** data and, optionally, **`ElementEntry`**s with latency and TCP retransmission values.



ELEMENT NAME	DATA TYPE	RANGE/EXAMPLE	DESCRIPTION
Ticks	DataTypes.INT	$-(2^{63})$ – 2^{63} - 1 | 235634	**Required**. Specifies the time set by the provider at the time the message was created.
RoundTripLatency	DataTypes.UINT	0 – 2^{63} - 1 | 358	Specifies the previous Round Trip Latency value (in microseconds) calculated by the provider.
TcpRetrans	DataTypes.UINT	0 – 2^{63} - 1 | 5	Specifies the current number of retransmissions.

Table 16: RTT Login `GenericMsg.Payload ElementList ElementEntrys`

3.4 Special Semantics

3.4.1 Authentication and Login Handling

Whether or not a user is authenticated depends on how the provider is implemented. However, providers are required to respond to Login Request Messages in the following manner:

- If the Login is accepted, the provider should send a Refresh message with **streamState = StreamStates.OPEN**, **dataState = DataStates.OK**, and **stateCode = StateCodes.NONE**.
- A login request can be rejected or closed by sending a Status message with **streamState = StreamStates.CLOSED** (or **StreamStates.CLOSED_RECOVER** if the user is allowed to attempt another login), **dataState = DataStates.SUSPECT**, and a **stateCode** that best describes the reason for the stream closure.
- If the provider closes the login stream, all other streams related to that login are implicitly closed without sending any item status messages to the consumer.
- If the consumer application closes the login stream using a close message, all streams related to that login are automatically closed without sending any item close messages to the provider.
- If a login stream is not open or was closed by either the consumer or the provider, the consumer must open a new Login stream before sending any other request messages. If the consumer application attempts to request data without an open Login stream, it will receive a status message indicating failure.
-

3.4.2 Negotiating msgKey.attrib Parameters

NOTE: The consumer must support all parameters specified by the provider in its Login response.

Several **`ReqMsg.MsgKey.Attrib`** parameters indicate how the provider should handle a connection including: **ProvidePermissionProfile**, **ProvidePermissionExpressions**, **SingleOpen**, **AllowSuspectData**, **InstanceId**, and **Role**. If the provider cannot support a specified value, it can change the value in the **`RespMsg.MsgKey.Attrib`**'s **`ElementList`** in its response to the consumer.

If the **`RespMsg.MsgKey.Attrib`** includes an **`ElementList`**, then the Elements should be checked to verify that the parameters have been accepted. If a request parameters differs from the response parameter, the consumer must support the parameters that the server provided in the **`RespMsg`**.

3.4.3 Group and Service Status

Group and service status messages do not apply to the Login domain.

3.4.4 Single Open and Allow Suspect Data Behavior

The **SingleOpen** and **AllowSuspectData** Elements that are passed via the **ReqMsg.MsgKey.Attrib** can effect how state information is processed. When the provider indicates support for SingleOpen behavior, the provider should drive the recovery of item streams. If no provider support is indicated, the consumer should drive any recovery.

The following table shows how a provider can convert messages to honor the consumer's **SingleOpen** and **AllowSuspectData** settings. The first column in the table shows the provider's actual **streamState** and **dataState**. Each subsequent column shows how this state information can be modified to follow that column's specific **SingleOpen** and **AllowSuspectData** settings. If any **SingleOpen** and **AllowSuspectData** configuration causes a contradiction in behavior (e.g., **SingleOpen** indicates that the provider should handle recovery, but **AllowSuspectData** indicates that the consumer does not want to receive suspect status), **SingleOpen** behavior takes precedence.

NOTE: The Enterprise Transport API does not perform any special processing based on the **SingleOpen** and **AllowSuspectData** settings. The provider application must perform any necessary conversion.

If **AcceptingRequests** is **FALSE**, new requests should not be made to a provider application, regardless of **ServiceState**. However, even if **AcceptingRequests** is **FALSE**, reissue requests can still be made for any item streams that are currently open to the provider.

The following table uses the abbreviations:

- SS for **streamState**
- DS for **dataState**

ACTUAL STATE INFORMATION	MESSAGE SENT WHEN: SINGLEOPEN = 1 ALLOWSUSPECTDATA = 1	MESSAGE SENT WHEN: SINGLEOPEN = 1 ALLOWSUSPECTDATA = 0	MESSAGE SENT WHEN: SINGLEOPEN = 0 ALLOWSUSPECTDATA = 1	MESSAGE SENT WHEN: SINGLEOPEN = 0 ALLOWSUSPECTDATA = 0
SS = OPEN **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = CLOSED_RECOVER **DS** = SUSPECT
SS = CLOSED_RECOVER **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = CLOSED_RECOVER **DS** = SUSPECT	**SS** = CLOSED_RECOVER **DS** = SUSPECT
New item request when[a]: **ServiceState** = DOWN **AcceptingRequests** = TRUE	**SS** = OPEN **DS** = SUSPECT	**SS** = OPEN **DS** = SUSPECT	**SS** = CLOSED_RECOVER **DS** = SUSPECT	**SS** = CLOSED_RECOVER **DS** = SUSPECT
New item requests when[a]: **ServiceState** = UP **AcceptingRequests** = TRUE	**SS** = OPEN **DS** = OK or SUSPECT based on individual item's state.	**SS** = OPEN **DS** = OK or SUSPECT based on individual item's state.	**SS** = OPEN **DS** = OK or SUSPECT based on individual item's state.	If **DS** == OK: **SS** = OPEN if **DS** == SUSPECT: **SS** = CLOSED_RECOVER

Table 17: SingleOpen and AllowSuspectData Handling

a. For more information, refer to Source Directory information in 4, Source Directory Domain.

3.5 Login Sample XML

3.5.1 Login Request Message Sample XML

```
<REQUEST domainType="LOGIN" streamId="1" containerType="NO_DATA" flags="0x04 (STREAMING)"
       dataSize="0">
   <key flags="0x26 (HAS_NAME|HAS_NAME_TYPE|HAS_ATTRIB)" name="patrick.kerkstra" nameType="1"
          attribContainerType="ELEMENT_LIST">
       <attrib>
          <elementList flags="0x08 (HAS_STANDARD_DATA)">
             <elementEntry name="ApplicationId" dataType="ASCII_STRING" data="256"/>
             <elementEntry name="ApplicationName" dataType="ASCII_STRING" data="Consumer"/>
             <elementEntry name="Position" dataType="ASCII_STRING" data="127.0.0.1/net"/>
             <elementEntry name="Password" dataType="ASCII_STRING" data="myPassword"/>
             <elementEntry name="ProvidePermissionProfile" dataType="UINT" data="1"/>
             <elementEntry name="ProvidePermissionExpressions" dataType="UINT" data="1"/>
             <elementEntry name="SingleOpen" dataType="UINT" data="1"/>
             <elementEntry name="AllowSuspectData" dataType="UINT" data="1"/>
             <elementEntry name="InstanceId" dataType="ASCII_STRING" data="myInstance"/>
             <elementEntry name="Role" dataType="UINT" data="0"/>
             <elementEntry name="DownloadConnectionConfig" dataType="UINT" data="0"/>
          </elementList>
       </attrib>
   </key>
   <dataBody>
   </dataBody>
</REQUEST>
```

Code Example 1: Login Request Message Sample XML Message Layout

3.5.2 Login Refresh Message Sample XML

```
<REFRESH domainType="LOGIN" streamId="1" containerType="NO_DATA" flags="0x168
      (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|CLEAR_CACHE)" groupId="0" State: Open/Ok/None - text:
      "Login accepted by host oaklrh196." dataSize="0">
   <key flags="0x26 (HAS_NAME|HAS_NAME_TYPE|HAS_ATTRIB)" name="patrick.kerkstra" nameType="1"
         attribContainerType="ELEMENT_LIST">
      <attrib>
         <elementList flags="0x08 (HAS_STANDARD_DATA)">
            <elementEntry name="AllowSuspectData" dataType="UINT" data="1"/>
            <elementEntry name="ApplicationId" dataType="ASCII_STRING" data="256"/>
            <elementEntry name="ApplicationName" dataType="ASCII_STRING" data="ADS"/>
            <elementEntry name="Position" dataType="ASCII_STRING" data="127.0.0.1/net"/>
            <elementEntry name="ProvidePermissionExpressions" dataType="UINT" data="1"/>
            <elementEntry name="ProvidePermissionProfile" dataType="UINT" data="0"/>
            <elementEntry name="SingleOpen" dataType="UINT" data="1"/>
            <elementEntry name="SupportOMMPost" dataType="UINT" data="1"/>
            <elementEntry name="SupportStandby" dataType="UINT" data="1"/>
            <elementEntry name="SupportBatchRequests" dataType="UINT" data="1"/>
            <elementEntry name="SupportViewRequests" dataType="UINT" data="1"/>
            <elementEntry name="SupportOptimizedPauseResume" dataType="UINT" data="1"/>
         </elementList>
      </attrib>
   </key>
   <dataBody>
   </dataBody>
</REFRESH>
```

Code Example 2: Login Refresh Message Sample XML Message Layout

4 Source Directory Domain

4.1 Description

The ***Source Directory*** domain model conveys:

- Information about all available services and their capabilities. This includes information about domain types supported within a service, the service's state, the quality of service, and any item group information associated with the service. Each service is associated with a unique **`serviceId`**. When requesting or responding to information associated with a specific service, the **`serviceId`** is specified in the **`msgKey`**.
- Status information associated with item groups. This allows a single message to change the state of all associated items, avoiding the need to send a status message for each individual item. The consumer is responsible for applying any changes to its open items. For details, refer to Section 4.3.1.2 and Section 4.3.1.3.
- Source Mirroring information between a Refinitiv Real-Time Advanced Data Hub and Open Message Model interactive provider applications exchanged via a specifically-formatted generic message as described in Section 4.2.5.

4.2 Usage

4.2.1 Source Directory Request Message

A Directory request message is encoded and sent by Open Message Model consumer applications. A consumer can request information about all services by omitting **serviceId** information, or specify a **serviceId** to request information about only that service. Because the Source Directory domain uses a **FilterList**, a consumer can indicate the specific source related information in which it is interested via a **msgKey.filter**. Each bit-value represented in the filter corresponds to an information set that can be provided in response messages. A consumer can change the requested filter via a reissue. For more details about the **FilterList** type, refer to the *Enterprise Transport API Java Edition Developers Guide*.

Refinitiv recommends that a consumer application minimally request Info, State, and Group filters for the Source Directory:

- The Info filter contains the **ServiceName** and **serviceId** data for all available services. When an appropriate service is discovered by the Open Message Model consumer, the **serviceId** associated with the service is used on subsequent requests to that service.
- The State filter contains status data for the service. Status data informs the Consumer whether the service is up (and available) or down (and unavailable).
- The Group filter conveys any item group status information, including group states and as regards the merging of groups if applicable.

NOTE: If an application does not subscribe to the Source Directory's group filter, it will not receive group status messages. This can result in potentially incorrect item state information, as relevant status information may be missed.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.SOURCE == 4**
qos	Not used.
worstQos	Not used.
priorityClass	Not used.
priorityCount	Not used.
extendedHeader	Not used.
msgKey.nameType	Not used.
msgKey.name	Not used.
msgKey.filter	**Required**. Specifies a filter indicating the specific data in which a consumer is interested. Available categories include: • **Directory.ServiceFilterFlags.INFO == 0x01** • **Directory.ServiceFilterFlags.STATE == 0x02** • **Directory.ServiceFilterFlags.GROUP == 0x04** • **Directory.ServiceFilterFlags.LOAD == 0x08** • **Directory.ServiceFilterFlags.DATA == 0x10** • **Directory.ServiceFilterFlags.LINK == 0x20** • **Directory.ServiceFilterFlags.SEQ_MCAST == 0x40** For details on the contents of each filter entry, refer to Section 4.3.1.1.

Table 18: Source Directory Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.serviceId	Optional. If a consumer wants information regarding only one particular service, it should specifythe **`msgKey.serviceId`** of that service (i.e., set a **`msgKey.flags`** value of **MsgKeyFlags.HAS_SERVICE_ID**). If the consumer wishes to receive information about all services, the consumer should not specify a **`msgKey.serviceId`** (i.e., the consumer should not set **MsgKeyFlags.HAS_SERVICE_ID**).
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 18: Source Directory Request Message Member Use (Continued)

4.2.2 Source Directory Refresh Message

A Directory Refresh Message is sent by Open Message Model provider and non-interactive provider applications. This message provides information about currently-known services, as well as additional details ranging from state information to provided domain types.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.SOURCE == 4**
state	**Required**. Indicates stream and data state information.
qos	Not used.
seqNum	Optional. A user-specified, item-level sequence number that the application can use to sequence messages within this stream.
groupId	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Not used.
msgKey.name	Not used.
msgKey.filter	**Required**. Identifies the filtered entries provided in this response. When possible, this should match the filter set in the consumer's request. For additional details, refer to the **`msgKey.filter`** member in Section 4.2.1.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The payload contains data about available services in the form of a **`Map`** where each entry's key is one **`serviceId`**. For additional details, refer to Section 4.3.1.

Table 19: Source Directory Refresh Message Member Use

4.2.3 Source Directory Update Message

A Source Directory Update Message is sent by Open Message Model provider and non-interactive provider applications. An Update message can:

- Indicate the addition or removal of services from the system or changes to existing services.
- Convey item group status information via the State and Group filter entries. For more information about item group use, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.SOURCE == 4**
updateType	Not used.
seqNum	Optional. A user-specified, item-level sequence number that the application can use to sequence messages in this stream.
conflationCount	Not used.
conflationTime	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Not used.
msgKey.name	Not used.
msgKey.filter	Optional. The **`msgKey.filter`** indicates which filter entries are provided in this response. For an update, this conveys only the ID values associated with filter entries present in the update payload. For more details, refer to the **`msgKey.filter`** member in Section 4.2.1.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The payload contains only the changed information associated with the provided services. For more details, refer to Section 4.3.1.

Table 20: Source Directory Update Message Member Use

4.2.4 Source Directory Status Message

A Source Directory status message is encoded and sent by both Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with a source directory stream.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.SOURCE == 4**
state	Optional. Contains stream and data state information for the directory stream.
groupId	Not used.
permData	Optional. If present, this is the new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
msgKey.serviceId	Not used.
msgKey.nameType	Not used.
msgKey.name	Not used.
msgKey.filter	**Required**. The filter represents the filter entries being provided in this response. When possible, this should match the filter as set in the consumer's request. For additional details, refer to the **`msgKey.filter`** member in Section 4.2.1
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 21: Source Directory Status Message Member Use

4.2.5 Source Directory Generic Message

A Source Directory Generic message is encoded and sent by a Refinitiv Real-Time Advanced Data Hub when using a 'hot standby' configuration. When running in hot standby mode, the Refinitiv Real-Time Advanced Data Hub can leverage source mirroring and use a generic message to convey usage information to upstream providers. A generic message can inform providers whether the Refinitiv Real-Time Advanced Data Hub is an active server without a standby (**ActiveNoStandby**), an active server with a standby (**ActiveWithStandby**) or a standby provider (**Standby**). This message is mainly for informational purposes, and allows a provider to better understand their role in a hot standby environment (the provider does not require a return action or acknowledgment).

A provider indicates each service's ability to process this message via the **`AcceptingConsumerStatus`** element in its Source Directory responses (refer to Section 4.3.1.1).

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.GENERIC == 7**
domainType	**Required**. **DomainTypes.SOURCE == 4**
partNum	Not used.
seqNum	Optional. A user-specified, item-level sequence number that the application can use to sequence messages in this stream.
secondarySeqNum	Not used.
permData	Not used.
extendedHeader	Not used.
msgKey.serviceId	Not used (**`Payload`** can contain information pertaining to multiple services, each of which is specifically identified in the payload).
msgKey.nameType	Not used.
msgKey.name	**Required**. The name of this message must be **`ConsumerStatus`**.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The payload is a **`Map`** whose entries contain the Source Mirroring status for each service. For the full structure, refer to Section 4.3.2.

Table 22: Source Directory Generic Message Member Use

4.3 Data

4.3.1 Source Directory Refresh and Update Payload

A list of services is represented by a **Map**. Each **MapEntry** represents a known service and is uniquely identified by its **msgKey.serviceId** (i.e., its key).

The information about each service is represented as a **FilterList**. Each **FilterEntry** contains one of six different categories of information. These categories should correspond to the **msgKey.filter** member of the refresh or update.

For example layouts of these messages, refer to Section 4.5.

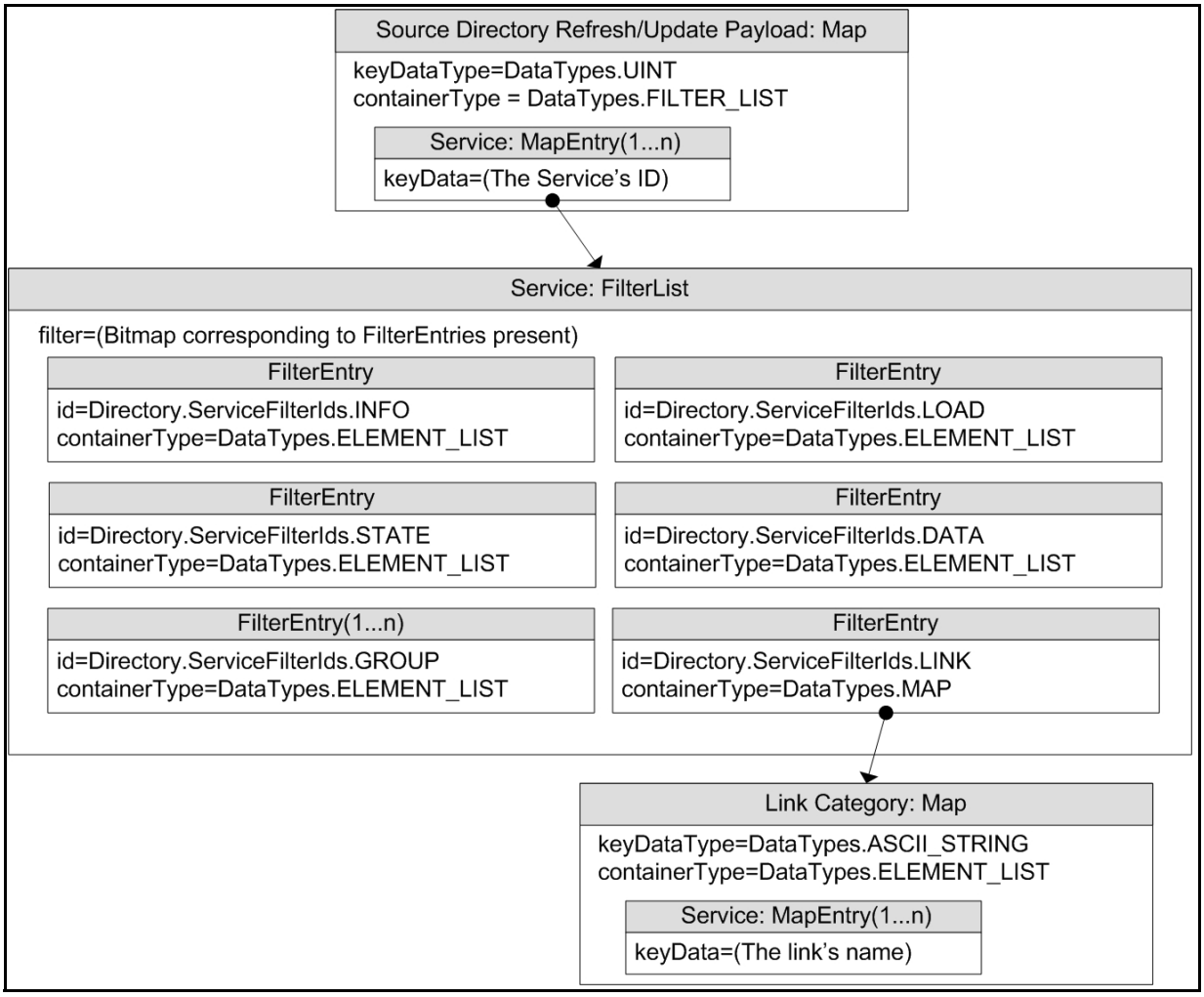


Figure 5. Source Directory Refresh and Update Message Payload

KEY TYPE	CONTAINER TYPE	PERMISSION DATA	DESCRIPTION
DataTypes.UINT	DataTypes.FILTER_LIST	Not used	Contains information for each known service. The key is the service's **msgKey.serviceId**.

Table 23: Source Directory Map Contents

There are six categories of information about a service, each represented by one **FilterEntry**. Categories can be added or updated in update messages (note that the clear action **FilterEntryActions.CLEAR** is not used, and that the Info category should not change).

FILTERENTRY ID (CORRESPONDING MSGKEY.FILTER BIT-VALUE)	TYPE	DESCRIPTION
Directory.ServiceFilterIds.INFO == 1 (Directory.ServiceFilterFlags.INFO == 0x01)	DataTypes.ELEMENT_LIST	**Provider applications must be able to provide this information.** Identifies a service and its available data. Refer to Section 4.3.1.1.
Directory.ServiceFilterIds.STATE = 2 (Directory.ServiceFilterFlags.STATE == 0x02)	DataTypes.ELEMENT_LIST	**Provider applications must be able to provide this information.** Describes the current state of a service (i.e., the service's current ability to provide data). Can also change the status of all items associated with this service. The effects of this category occur immediately. Therefore, the initiating **`UpdateMsg`** should set **UpdateMsgFlags.DO_NOT_CONFLATE**. Refer to Section 4.3.1.2.
Directory.ServiceFilterIds.GROUP == 3 (Directory.ServiceFilterFlags.GROUP == 0x04)	DataTypes.ELEMENT_LIST	Manages group information. Can change the status of a group of items or merge items from one group to another. The effects of this category occur immediately and only affect existing items. Therefore, the initiating **`UpdateMsg`** should set **UpdateMsgFlags.DO_NOT_CONFLATE** and **UpdateMsgFlags.DO_NOT_CACHE**. Refer to Section 4.3.1.3.
Directory.ServiceFilterIds.LOAD == 4 (Directory.ServiceFilterFlags.LOAD == 0x08)	DataTypes.ELEMENT_LIST	Information about the current workload of this service, including how many items are currently being serviced. Optionally, the initiating **`UpdateMsg`** can set **UpdateMsgFlags.DO_NOT_CONFLATE**. Refer to Section 4.3.1.4.
Directory.ServiceFilterIds.DATA == 5 (Directory.ServiceFilterFlags.DATA == 0x10)	DataTypes.ELEMENT_LIST	Includes broadcast data that applies to all items requested from that service. This information is typically provided in a dedicated **`UpdateMsg`** and sent independently of other filter entries. The data filter is commonly used with ANSI Page-based data. Flag values **UpdateMsgFlags.DO_NOT_CONFLATE** and **UpdateMsgFlags.DO_NOT_CACHE** can optionally be set to prevent conflation and caching of this content. Refer to Section 4.3.1.5.

Table 24: Source Directory `MapEntry` Filter Entries

FILTERENTRY ID (CORRESPONDING MSGKEY.FILTER BIT-VALUE)	TYPE	DESCRIPTION
Directory.ServiceFilterIds.LINK = 6 (Directory.ServiceFilterFlags.LINK = 0x20)	DataTypes.MAP	Provides information about individual upstream sources that provide data for this service. This is primarily used by systems that aggregate sources (such as the Refinitiv Real-Time Advanced Data Hub) for identification and load balancing, and is not required to be processed by a consumer application. Refer to Section 4.3.1.6.
Directory.ServiceFilterIds.SEQ_MCAST = 7 (Directory.ServiceFilterFlags.SEQ_MCAST = 0x40)	DataTypes.ELEMENT_LIST	Provides information about the EDF (Elektron Direct Feed) and EDF connection information. For further information, refer to the *Elektron Direct Feed Development Guide*.

Table 24: Source Directory `MapEntry` Filter Entries (Continued)

4.3.1.1 Source Directory Info Filter Entry

The Info filter entry (**Directory.ServiceFilterFlags.INFO**, **Directory.ServiceFilterIds.INFO**) conveys information that identifies a service and the content it can provide. This includes information about provided domain types (e.g., Market Price, Market By Order), available QoS, and the names of any dictionaries required to parse the published content.

The Info **`FilterEntry`** should be present when a service is first added, and should not be changed as long as the service remains in the list.

NOTE: The Refinitiv Real-Time Advanced Data Hub does not track services that are brought down. If you bring up a service after having brought it down, you must again include the Info filter entry.

If a FilterEntry element uses a default value, it is included in the element's description.

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
Name	DataTypes.ASCII_STRING	e.g., IDN_RDF	**Required.** Specifies the service's name. This value allows for mapping between the service name and the **`serviceId`**.
Vendor	DataTypes.ASCII_STRING	e.g., Refinitiv	Specifies the name of the vendor that provides the data for this service.
IsSource	DataTypes.UINT	0 | 1	Specifies whether the service aggregates content from multiple sources. Available values are: • **0**: The service aggregates multiple sources into a single service. This is the default behavior. • **1**: The service is provided directly by the original publisher

Table 25: Source Directory Info Filter Entry Elements

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
Capabilities	DataTypes.ARRAY of DataTypes.UINT	e.g., [5, 6]	**Required.** Lists the domains which this service can provide. For example, a list containing **DomainTypes.DICTIONARY** (**5**) and **DomainTypes.MARKET_PRICE** (**6**) indicates a consumer can request dictionaries and Market Price data from this service. Set **`Array.itemLength`** to **1**, as each **`domainType`** uses only one byte.
DictionariesProvided	DataTypes.ARRAY of DataTypes.ASCII_STRING	e.g., RWFFld	Lists the Dictionary names that this service can provide. A consumer can obtain these dictionaries by requesting them by name on the **DomainTypes.DICTIONARY** domain. For details, refer to 5, Dictionary Domain.
DictionariesUsed	DataTypes.ARRAY of DataTypes.ASCII_STRING	e.g., RWFFld, RWFEnum	**Conditional.** Lists the Dictionary names that might be required to fully process data from this service. Whether or not the dictionary is required depends on the consumer's needs. For example: if the consumer is not a display application, it might not need an Enumerated Types Dictionary. For details, refer to 5, Dictionary Domain.
QoS	DataTypes.ARRAY of DataTypes.QOS	e.g., Realtime, Tick-By-Tick	Specifies the available Qualities of Service (QoS). • If the data comes from one source, there will usually be only one QoS. • If there are multiple sources, more than one QoS may be available. The default QoS is **Realtime, Tick-By-Tick**. Thus. if a QoS is not provided, or if the Transport API receives a QoS value of **Unspecified**, the Transport API assumes the service provides a QoS of **Realtime**, **Tick-By-Tick**. For more information about QoS use and handling, refer to the *Enterprise Transport API Java Edition Developers Guide*.
SupportsQoSRange	DataTypes.UINT	0 | 1	Indicates whether the provider supports a QoS range when requesting an item. If supported, a consumer can indicate an acceptable range via the **`qos`** and **`worstQos`** members of an **`RequestMsg`**. • **0**: The provider does not support QoS range requests. This is the default behavior. • **1**: The provider supports QoS range requests.
ItemList	DataTypes.ASCII_STRING		Specifies the name of a SymbolList (i.e., a specific item requested to get the names of all items available for this service). The consumer requests this item via the **DomainTypes.SYMBOL_LIST** domain (See 11, Symbol List Domain).

Table 25: Source Directory Info Filter Entry Elements (Continued)

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
SupportsOutOfBandSnapshots	DataTypes.UINT	0 | 1	Indicates whether Snapshot requests can still be made after reaching the **OpenLimit** (refer to Section 4.3.1.4). • **0**: Snapshot requests cannot be made if the **OpenLimit** is reached. • **1**: Snapshot requests can be made even when the **OpenLimit** is reached. This is the default behavior.
AcceptingConsumerStatus	DataTypes.UINT	0 | 1	Indicates whether a service can accept and process messages related to Source Mirroring (refer to Section 4.2.4). • **0**: The service cannot accept and process messages related to Source Mirroring. • **1**: The service can accept and process messages related to Source Mirroring. This is the default behavior.

Table 25: Source Directory Info Filter Entry Elements (Continued)

4.3.1.2 Source Directory State Filter Entry

The State filter entry (**Directory.ServiceFilterFlags.STATE**, **Directory.ServiceFilterIds.STATE**) conveys information about the current state of a service. This information usually has some bearing on the availability of data from a service. If a service becomes temporarily unavailable or becomes available again, consumers are informed via updates to this category.

A State filter entry should be present in the initial refresh, and then updated whenever needed.

NOTE: The Refinitiv Real-Time Advanced Data Hub does not track services that are brought down. If you bring up a service after having brought it down, you must include the Info filter entry (refer to Section 4.3.1.1).

The Status element can change the state of items provided by this service. Prior to changing a service status, Refinitiv recommends that you issue item or group status messages to update item states. For example, before bringing down a service, a provider application should change the **Status** element of all items to **StreamStates.CLOSED_RECOVER**.

Any default behavior is explained in the Element's description.

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
ServiceState	DataTypes.UINT	0 | 1	**Required.** Indicates whether the original provider of the data is available to respond to new requests. If the service is down, requests for data may be handled by the immediate upstream provider (to which the consumer is directly connected). However, because the most current data might be serviced from a cached copy while the source is down, the most current data may not be immediately available. Changes to **ServiceState** do not affect streams that are already open. Available values are: • **0**: Service is Down • **1**: Service is Up Refer to Section 4.4.2.
AcceptingRequests	DataTypes.UINT	0 | 1	Indicates whether the immediate provider can accept new requests and/or handle reissue requests on already open streams. Existing streams remain unaffected, however new requests may be rejected. **`AcceptingRequests`** defaults to **1**. Available values are: • **0**: The provider cannot accept new requests on existing streams. • **1**: The provider can accept new requests on existing streams. Refer to Section 4.4.2.
Status	DataTypes.STATE	e.g., [StreamStates.OPEN, DataStates.OK, StateCodes.NONE]	Specifies a status change to apply to all items provided by this service. It is equivalent to sending a **`StatusMsg`** for each item. The **`streamState`** is only allowed to be **StreamStates.OPEN** or **StreamStates.CLOSED_RECOVER**. This status only applies to item streams that have received a refresh or status of **OPEN**/**OK**. Refer to Section 4.4.3.1.

Table 26: Source Directory State `FilterEntry` Elements

4.3.1.3 Source Directory Group Filter Entry

The Group filter entry (**Directory.ServiceFilterFlags.GROUP**, **Directory.ServiceFilterIds.GROUP**) conveys item group status and item group merge information. Every item stream is associated with an item group as defined by the **groupId** provided with the item's **RefreshMsg** or **StatusMsg**. If some kind of change impacts all items within the same group, only a single group status message need be provided. For more information on item group use and handling, see the *Enterprise Transport API Java Edition Developers Guide*.

If multiple group **FilterEntry**s are received in a single **FilterList**, then they should be applied in the order in which they were received.

Any default behavior is explained in the Element's description.

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Group	DataTypes.BUFFER	e.g., 1.26.102	**Required**. Specifies the **groupId** with which this information is associated. This is typically represented as a series of 2-byte unsigned integers (i.e., two-byte unsigned integers written directly next to each other in the buffer). The example provided in the **RANGE** / **EXAMPLE** column of this table shows such a series, with inserted dots to help indicate two-byte value. When encoded into a buffer, do not include these dots.
MergedToGroup	DataTypes.BUFFER	e.g., 1.26.110	Changes all items whose group currently matches the **Group** element to the specified **MergedToGroup**. **NOTE:** The consumer should change the **groupId** of those items to match this element.
Status	DataTypes.STATE	e.g., [StreamStates.OPEN, DataStates.OK, StateCodes.NONE,]	A status change to be applied to all items whose **groupId** matches the **Group** element. It is equivalent to sending a **StatusMsg** to each item. • The **streamState** is only allowed to be **StreamStates.OPEN** or **StreamStates.CLOSED_RECOVER**. • If you need to convey group status **text** or **code** information without changing the data state, use the value **DataStates.NO_CHANGE**. • If present in the same message as a **MergedToGroup** element, this change should be applied before the merge. This change only applies to item streams that have received a refresh or status with a state of **OPEN**/**OK**. Refer to Section 4.4.3.2

Table 27: Source Directory Group FilterEntry Elements

4.3.1.4 Source Directory Load Filter Entry

The Load filter entry (**Directory.ServiceFilterFlags.LOAD**, **Directory.ServiceFilterIds.LOAD**) conveys information about the service's workload. If multiple services can provide desired data, a consumer can use service workload information to help decide which to use. None of these elements are required, nor have a default value.

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
OpenLimit	DataTypes.UINT	0 – 4,294,967,295	Maximum number of streaming items that the client is allowed to open for this service. If the service supports out-of-band snapshots, snapshot requests do not count against this limit (refer to Section 4.3.1.1).
OpenWindow	DataTypes.UINT	0 - 4,294,967,295	Maximum number of outstanding requests (i.e., requests for items not yet open) that the service will allow at any given time. If **OpenWindow** is **0**, the behavior is the same as setting **AcceptingRequests** to **0** and no open item request is accepted. The provider should not assume that the **OpenWindow** becomes effective immediately.
LoadFactor	DataTypes.UINT	0-65,535	A number indicating the current workload on the source providing the data. This number and the means of its calculation vary based on the system (i.e., bandwidth usage, CPU usage, number of clients, etc). The only requirements are that: • The **LoadFactor** should be calculated the same way for all services in a system. • A more heavily-loaded service should have a higher **LoadFactor** than one that is less loaded.

Table 28: Source Directory Load `FilterEntry` Elements

4.3.1.5 Source Directory Data Filter Entry

The Data filter entry (**Directory.ServiceFilterFlags.DATA**, **Directory.ServiceFilterIds.DATA**) conveys information that should be applied to all items associated with the service. This is commonly used for services that provide ANSI Page-based data. These elements has do not have a default value.

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Type	DataTypes.UINT	• Time(1) • Alert (2) • Headline (3) • Status (4) • Reserved values : 0 - 1023	**Conditional**. You must include **`Type`** when data is present. Explains the content of the **`Data`**.
Data	Any Data Type		Data that should be applied to all items from the service; commonly used for services providing ANSI Page-based data. The contents of this element should be applied as an update to every item open for this stream. After the data fans out, it does not need to be cached as part of the source directory.

Table 29: Source Directory Data `FilterEntry` Elements

4.3.1.6 Source Directory Link Filter Entry

The Link filter entry (**Directory.ServiceFilterFlags.LINK**, **Directory.ServiceFilterIds.LINK**) conveys information about the upstream sources that provide data to a service.

This information is represented as a **`Map`**, where each **`MapEntry`** represents one upstream source. The map entry key is the name associated with the communication link, and is of type **DataTypes.ASCII_STRING**. This name is scoped globally, and if multiple sources have the same name, they are assumed to be identical and the aggregating system will balance requests among them.

A Refinitiv Real-Time Advanced Data Hub component can leverage this information for failover and hot standby functionality. More detailed information is available in the Refinitiv Real-Time Advanced Data Hub documentation. A typical consumer application can treat this entry as mainly informational. The consumer should use the State **`FilterEntry`** to make programmatic decisions about service availability and status.

Any default behavior is explained in the Element's description.

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Type	DataTypes.UINT	1 | 2	Indicates whether the upstream source is interactive or broadcast. This does not describe whether the service itself is interactive or broadcast. • **1**: The upstream source is interactive (this is the default). • **2**: The upstream source is a broadcast source.
LinkState	DataTypes.UINT	0 | 1	**Required.** Indicates whether the upstream source is up or down • **0**: The upstream source is down. • **1**: The upstream source is up.
LinkCode	DataTypes.UINT	0 - 3	Provides additional status information about the upstream source. • **0**: None (this is the default) • **1**: Ok • **2**: RecoveryStarted • **3**: RecoveryCompleted
Text	DataTypes.ASCII_STRING	N/A	Explains the **`LinkState`** and **`LinkCode`**. Text defaults to "".

Table 30: Source Directory Link `FilterEntry` Map Contents

4.3.1.7 Source Directory Sequenced Multicast Filter Entry

The Sequenced Multicast filter entry (**Directory.ServiceFilterFlags.SEQ_MCAST**, **Directory.ServiceFilterIds.SEQ_MCAST**) convey information about EDF components and connection information.

This information is represented as an **`ElementList`**, where each **`ElementEntry`** represents information about the EDF component. The entries that contain the information about the Real Time Streams or Gap Fill Servers will have **`ElementEntry`** that contain **`Vector`** with **`VectorEntry`** with additional connection information (Multicast Group, Port, and Domain).

While none of the **`FieldEntry`** Map elements are required, all of the **`FieldEntry`** vector elements are required.

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
ReferenceDataServerHost	DataTypes.ASCII_STRING	10.0.1.125	The hostname, or IP address, of the Reference Data Server.
ReferenceDataServerPort	DataTypes.UINT	14000	The port number to which the Reference Data Server is bound and on which it listens for incoming connections.
SnapshotServerHost	DataTypes.ASCII_STRING	10.0.1.125	The hostname, or IP address, of the Snapshot server.
SnapshotServerPort	DataTypes.UINT	14002	The port number to which the Snapshot Server is bound and on which it listens for incoming connections.
GapRecoveryServerHost	DataTypes.ASCII_STRING	10.0.1.125	The hostname, or IP address, of the Gap Recovery Server.
GapRecoveryServerPort	DataTypes.UINT	14001	The port number to which the Gap Recovery Server is bound and on which it listens for incoming connections.
StreamingMulticastChannels	DataTypes.VECTOR of DataTypes.ELEMENT_LIST		Multicast channel/port information for the streaming data provided by the service.
GapMulticastChannels	DataTypes.VECTOR of DataTypes.ELEMENT_LIST		Multicast channel/port information used by the Gap Recovery Server.

Table 31: Source Directory Sequenced Multicast `FilterEntry` Map Contents

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
MulticastGroup	DataTypes.ASCII_STRING	224.1.62.2	**Required.** The Multicast channel used.
Port	DataTypes.UINT	30001	**Required.** The port used.
Domain	DataTypes.UINT	6 (MarketPrice)	**Required.** The domain covered by this multicast channel and port.

Table 32: Source Directory Sequenced Multicast `FilterEntry` Vector Contents

4.3.2 Source Directory ConsumerStatus Generic Message Payload

NOTE: **GenericMsg**(s) are supported for the **DIRECTORY** Refinitiv Domain Model only for sending / receiving information related to ConsumerStatus/Source Mirroring Mode.

The data structure for the ConsumerStatus message is a **Map**. Each **MapEntry** sends status to one service and is uniquely identified by **msgKey.serviceId** (its key). Each entry contains an **ElementList** with one **ElementEntry** that indicates how the provider is used. **MapEntry**s do not use permission data.

KEY TYPE	VALUE TYPE	DESCRIPTION
DataTypes.UINT	DataTypes.ELEMENT_LIST	**Required**. Represents a service in the Source Directory. Contains an **ElementList** with Source Mirroring information for that service.

Table 33: Source Directory GenericMsg MapEntry

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
SourceMirroringMode	DataTypes.UINT	0 - 2	**Required**. Indicates how the downstream component uses the service. There is no default setting. **SourceMirroringMode** can have any of the following values: • **0**: ActiveNoStandby. The downstream device uses the data from this service, and does not receive it from any other service. • **1**: ActiveWithStandby. The downstream device uses the data from this service, but also receives it from another service. • **2**: Standby. The downstream device receives data from this service, but actually uses data from another service. A reply from the provider application is not needed because this is for informational use only.

Table 34: Source Directory Generic Message MapEntry Elements

4.4 Special Semantics

4.4.1 Multiple Streams

Unlike other MessageModelTypes, two directory streams can be open with identical message key information. It is also permissible to change an open stream's filter.

4.4.2 ServiceState and AcceptingRequests

The **ServiceState** and **AcceptingRequests** elements in the State filter entry work together to indicate the ability of a particular service to provide data:

- **ServiceState** indicates whether the source of the data is accepting requests.
- **AcceptingRequests** indicates whether the immediate upstream provider (the provider to which the consumer is directly connected) can accept new requests. If **False**, new requests are rejected while existing streams remain unaffected (reissue requests can still be made for any item streams that are currently open to the provider).

The following table applies only to new requests. Changes to the status of current streams should be sent using individual item status messages or group status messages via the Group filter entry (refer to Section 4.3.1.3).

SERVICESTATE	ACCEPTINGREQUESTS	MEANING
Up(**1**)	Yes (**1**)	New requests and reissue requests can be successfully processed.
Up(**1**)	No (**0**)	Although the source of data is available, the immediate provider is not accepting new requests. It may be possible to request from another provider. However, reissue requests on already open streams can be processed.
Down (**0**)	Yes (**1**)	The source of data is not available. The immediate provider, however, can accept the request and forward it when the source becomes available.
Down (**0**)	No (**0**)	Neither the source nor the immediate provider is accepting new requests.

Table 35: ServiceState and AcceptingRequests

4.4.3 Service and Group Status Values

The **Status** elements in the State and Group FilterEntries are transient. Their values should be applied to all existing streams. The values should not be cached and should not affect any new requests.

4.4.3.1 Service Status

Providers can use a directory's **ServiceState.Status** element to efficiently change the state of all of a service's existing streams with a single message. The **ServiceState.Status** does not apply to requests that are currently pending a first refresh or status response message. Enterprise Message API consumer implementation normally fans out state from the Status Element to all items associated with the service. When Enterprise Message API does this, it will not forward this Element to the application. Instead, the application receives a **`StatusMsg`** for each item from the service. The other elements from the **ServiceState** FilterEntry will still be sent to the application.

4.4.3.2 Group Status

The Group FilterEntry can be used to efficiently change the state of a large number of items with a single message. The **Group.Status** does not apply to requests that are currently pending a first refresh or status response message. Enterprise Message API consumer implementation normally fans out group messages to all items associated with the group. When Enterprise Message API does this, it will not forward this FilterEntry to the application. Instead, the application will receive a **`StatusMsg`** for each item in the group.

4.4.4 Removing a Service

If a provider needs to remove a service from the list of known services, it should send the service's **`MapEntry`** with the action set to **MapEntryActions.DELETE**. A consumer should place all open items associated with this service in the **StreamStates.CLOSED_RECOVER**.

All services associated with a Source Directory stream are removed if:

- The connection between the provider and consumer is closed or lost
- The provider sends a state of **StreamStates.CLOSED** or **StreamStates.CLOSED_RECOVER** on the Source Directory stream.
- The provider sends a message with the **RefreshMsgFlags.CLEAR_CACHE** on a **`RefreshMsg`** or **StatusMsgFlags.CLEAR_CACHE** on a **`StatusMsg`** on the Source Directory stream.

NOTE: Though not best practice, some applications may continue to store service information, even after a service is removed. If this is the case, the application should advertise the service as **Down** and not accepting requests.

4.4.5 Non-existent Services

If no services currently exist or the consumer issued a Source Directory request and specified an unknown **`serviceId`**, a Source Directory refresh should be sent containing an empty **`Map`**. The provider can then add services, via Source Directory updates, as they become available.

4.5 Source Directory Sample XML

4.5.1 Source Directory Request Message Sample XML

```
<REQUEST domainType="SOURCE" streamId="2" containerType="NO_DATA" flags="0x04 (STREAMING)"
       dataSize="0">
   <key flags="0x08 (HAS_FILTER)" filter="63"/>
   <dataBody>
   </dataBody>
</REQUEST>
```

Code Example 3: Source Directory Request Message Sample XML Message Layout

4.5.2 Source Directory Refresh Message Sample XML

```
<REFRESH domainType="SOURCE" streamId="2" containerType="MAP" flags="0x168
       (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|CLEAR_CACHE)" groupId="0" State: Open/Ok/None -
       text: "" dataSize="475">
   <key flags="0x08 (HAS_FILTER)" filter="63"/>
   <dataBody>
      <map flags="0x00" countHint="0" keyPrimitiveType="UINT" containerType="FILTER_LIST" >
         <mapEntry flags="0x00" action="ADD" key="1257" >
            <filterList containerType="ELEMENT_LIST" countHint="0" flags="0x00">
               <filterEntry id="1" action="SET" flags="0x02" containerType="ELEMENT_LIST">
                  <elementList flags="0x08 (HAS_STANDARD_DATA)">
                     <elementEntry name="Name" dataType="ASCII_STRING" data="MY_SERVICE"/>
                     <elementEntry name="SupportsQoSRange" dataType="UINT" data="0"/>
                     <elementEntry name="Capabilities" dataType="ARRAY">
                        <array itemLength="1" primitiveType="UINT">
                           <arrayEntry data="5"/>
                           <arrayEntry data="6"/>
                        </array>
                     </elementEntry>
                     <elementEntry name="QoS" dataType="ARRAY">
                        <array itemLength="0" primitiveType="QOS">
                           <arrayEntry Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0/>
                        </array>
                     </elementEntry>
                     <elementEntry name="DictionariesProvided" dataType="ARRAY">
                        <array itemLength="0" primitiveType="ASCII_STRING">
                           <arrayEntry data="RWFFld"/>
                           <arrayEntry data="RWFEnum"/>
                        </array>
                     </elementEntry>
                     <elementEntry name="DictionariesUsed" dataType="ARRAY">
                        <array itemLength="0" primitiveType="ASCII_STRING">
                           <arrayEntry data="RWFFld"/>
```

```
                              <arrayEntry data="RWFEnum"/>
                           </array>
                        </elementEntry>
                        <elementEntry name="Vendor" dataType="ASCII_STRING" data="MY_VENDOR"/>
                        <elementEntry name="IsSource" dataType="UINT" data="1"/>
                     </elementList>
                  </filterEntry>
                  <filterEntry id="2" action="SET" flags="0x02" containerType="ELEMENT_LIST">
                     <elementList flags="0x08 (HAS_STANDARD_DATA)">
                        <elementEntry name="ServiceState" dataType="UINT" data="1"/>
                        <elementEntry name="AcceptingRequests" dataType="UINT" data="1"/>
                     </elementList>
                  </filterEntry>
                  <filterEntry id="4" action="SET" flags="0x02" containerType="ELEMENT_LIST">
                     <elementList flags="0x08 (HAS_STANDARD_DATA)">
                        <elementEntry name="OpenLimit" dataType="UINT" data="50000"/>
                     </elementList>
                  </filterEntry>
               </filterList>
            </mapEntry>
            <!-- Additional entries... -->
         </map>
      </dataBody>
</REFRESH>
```

Code Example 4: Source Directory Refresh Message Sample XML Message Layout

5 Dictionary Domain

5.1 Description

NOTE: **`GenericMsg`**(s) are not supported for the Dictionary domain model.

The Open Message Model can optimize bandwidth usage by reducing or removing the need to constantly communicate well-known information (e.g., names and data types associated with information in a **`FieldList`**). Using these techniques, information is instead contained in a field dictionary, where the field list contains only **`fieldId`** references to information in the dictionary.

A provider application can indicate any dictionaries needed to parse published content. To reconstruct omitted information, consumer applications reference required dictionaries when decoding. Dictionaries may be available locally (i.e., in a file) or available for request over the network from an upstream provider.

The following dictionaries provide domain models for network requests:

- ***Field Dictionary***: Stores data referenced by the **`FieldList`**. Each **`fieldId`** in a **`FieldEntry`** corresponds to an entry in the Field Dictionary, which provides information such as the field's name (e.g., **BID**) and data type (e.g., **DataTypes.INT**). Additional information (such as rippling fields and expected cache-sizing requirements) are also present.
- ***Enumerated Types Dictionary***: Contains tables defining values for enumerated values of type **`Enum`** (**DataTypes.ENUM**). Each table indicates the **`fieldId`** values of all fields that use the data in the table, as well as the possible enumerated values. For example, a field indicating the currency of an item will use a table listing enumerations of various currencies. If a consumer decodes the value of that field (e.g., **840**), it can cross reference that value with its copy of the table. The entry the consumer finds will contain a string that the consumer can print (e.g. **USD**), and possibly a more meaningful description as well.
- ***Global Field Set Definition*** and ***Global Element Set Definition***: Provides global set definition database information. These databases contain several set definitions, each of which contains a list of **`fieldId`** (for field lists) or **`Name`** (for element lists) and value pairs. These definitions are used with structured data and provide bandwidth savings by not sending data contained in the set definition on the wire.

The consumer should store the information contained in these dictionaries and may need to refer to them when decoding data. For assistance, the Transport API provides utility functions for loading, encoding, and decoding the Refinitiv Field Dictionary (**RDMFieldDictionary**) and Enumerated Types Dictionaries (**enumtype.def**) (for details, refer to Section 5.12).

5.2 Decoding Field List Contents with Field and Enumerated Types Dictionaries

By itself, a **FieldEntry** contains only the **fieldId** and its associated encoded value in **encodedData**. With few exceptions, the type information is not sent with the data and will appear as **DataTypes.UNKNOWN**. To decode the value, the application must cross reference the **fieldId** with the correct Field Dictionary to determine its type. The **FieldList.dictionaryId** can optionally convey the identifier of the associated dictionary.

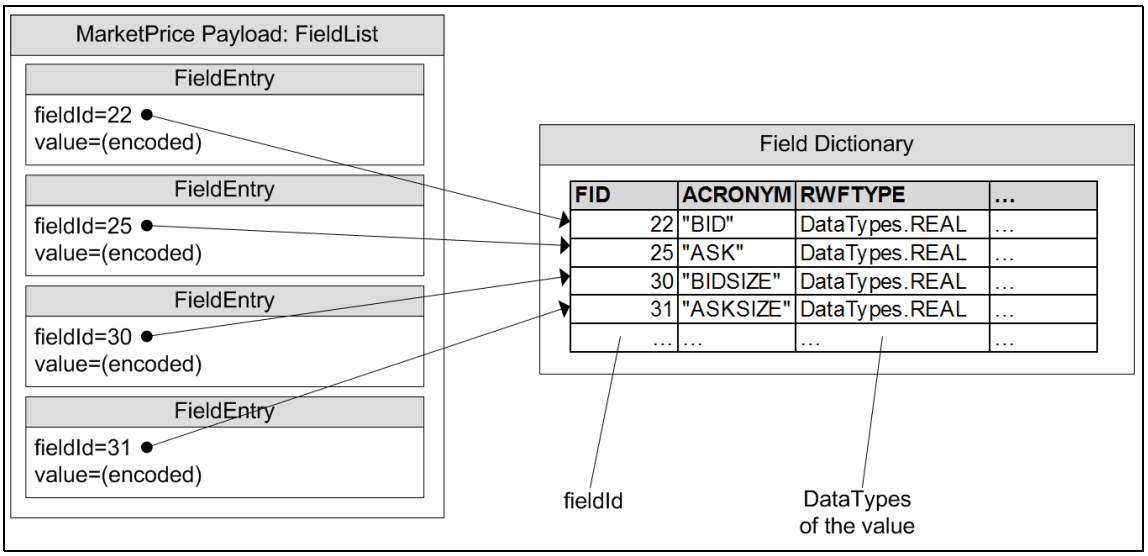


Figure 6. FieldList Referencing Field Dictionary

The consumer matches the **fieldId** of the **FieldEntry** to the information about that Field IDentifier in the dictionary. This tells the consumer the type (**DataTypes**) of the data contained in the **FieldEntry**. The consumer can now decode the data by calling the appropriate decode function for that type.

If the field's type is **DataTypes.ENUM**, the application expects and looks for a table of values in a corresponding Enumerated Types Dictionary.

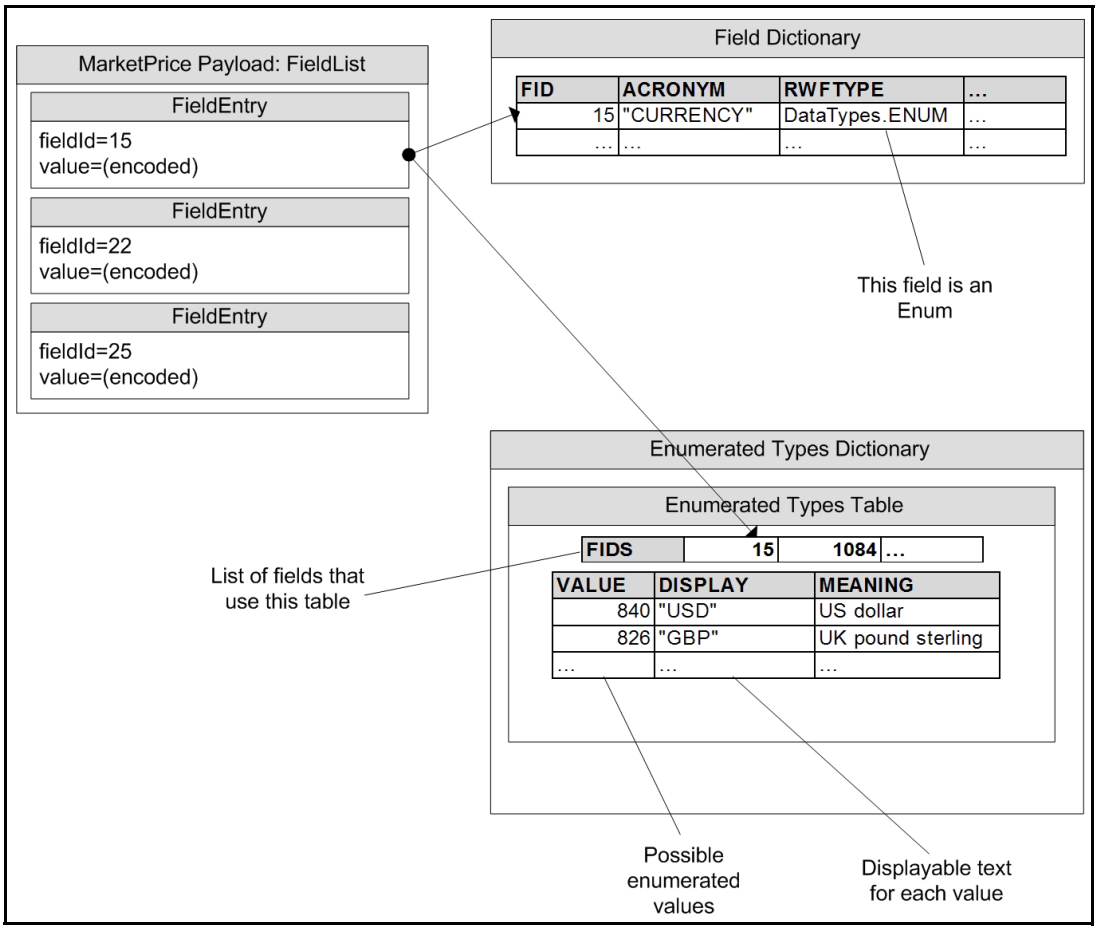


Figure 7. `FieldEntry` Referencing an Enumerated Types Table

The consumer, having decoded the enumerated value (e.g., **840**), finds the correct table that defines the field and looks up the enumerated value in that table. The value will have a displayable string associated with it (e.g., **USD**).

5.3 Usage

5.3.1 Dictionary Request Message

A dictionary request message is encoded and sent by Open Message Model consumer applications. The request indicates the name of the desired dictionary and how much information from that dictionary is needed.

Though updates are not sent on dictionary streams, Refinitiv recommends that the consumer make a streaming request (setting **RequestMsgFlags.STREAMING**) so that it is notified whenever the dictionary version changes.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.DICTIONARY == 5**
qos	Not used.
worstQos	Not used.

Table 36: Dictionary Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
priorityClass	Not used.
priorityCount	Not used.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the **`msgKey.serviceId`** of the service from which the consumer requests the dictionary.
msgKey.nameType	Not used.
msgKey.name	**Required**. Specify a **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME**. Specifies the **`msgKey.name`** of the desired dictionary as seen in the Source Directory response (refer to Section 4.3.1.1).
msgKey.filter	**Required**. The filter represents the desired verbosity of the dictionary. The consumer should set the **`msgKey.filter`** according to how much information is needed: • **Dictionary.VerbosityValues.INFO == 0x00**: Provides version information only. • **Dictionary.VerbosityValues.MINIMAL == 0x03**: Provides information needed for caching. • **Dictionary.VerbosityValues.NORMAL == 0x07**: Provides all information needed for decoding. • **Dictionary.VerbosityValues.VERBOSE == 0x0F**: Provides all information (including comments). Providers are not required to support the **MINIMAL** and **VERBOSE** filters.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 36: Dictionary Request Message Member Use (Continued)

5.3.2 Dictionary Refresh Message

A dictionary request message is encoded and sent by Open Message Model consumer applications. The request indicates the name of the desired dictionary and how much information from that dictionary is needed.

Though updates are not sent on dictionary streams, Refinitiv recommends that the consumer make a streaming request (setting **RequestMsgFlags.STREAMING**) so that it is notified whenever the dictionary version changes.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.DICTIONARY == 5**
qos	Not used.
worstQos	Not used.
priorityClass	Not used.
priorityCount	Not used.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the **`msgKey.serviceId`** of the service from which the consumer requests the dictionary.
msgKey.nameType	Not used.
msgKey.name	**Required**. Specify a **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME**. Specifies the **`msgKey.name`** of the desired dictionary as seen in the Source Directory response (refer to Section 4.3.1.1).
msgKey.filter	**Required**. The filter represents the desired verbosity of the dictionary. The consumer should set the **`msgKey.filter`** according to how much information is needed: • **Dictionary.VerbosityValues.INFO == 0x00**: Provides version information only. • **Dictionary.VerbosityValues.MINIMAL == 0x03**: Provides information needed for caching. • **Dictionary.VerbosityValues.NORMAL == 0x07**: Provides all information needed for decoding. • **Dictionary.VerbosityValues.VERBOSE == 0x0F**: Provides all information (including comments). Providers are not required to support the **MINIMAL** and **VERBOSE** filters.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 37: Dictionary Request Message Member Use

5.3.3 Dictionary Status Message

A dictionary status message is encoded and sent by Open Message Model Interactive and non-interactive provider applications. This message can indicate changes to a dictionary's version.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.DICTIONARY == 5**
state	Optional. Used to indicate a change to a dictionary version. For more details, refer to Section 5.9.3.
groupId	Not used.
permData	**Conditional**. Used if the provided dictionary requires permissioning.
extendedHeader	Not used.
msgKey.serviceId	Optional.
msgKey.nameType	Not used.
msgKey.name	Optional. Should match name that was requested.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 38: Dictionary Status Message Member Use

5.4 Payload and Summary Data for the Refresh Message

For example layouts of some dictionary messages, refer to Section 5.11.

The payload varies depending on the type of dictionary being sent. The domain model layout for each type of dictionary is described in the following sections.

Although the structure varies for each type of dictionary, each uses an **`ElementList`** to identify its type, version, and **`DictionaryId`**. If sending the dictionary as a multi-part refresh, this data must be present only in the first part. Any default behavior is included in the dictionary type's description.

NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Version	DataTypes.ASCII_STRING	"1.1"	**Required**. Specifies the version of the provided dictionary. For additional details on dictionary versions, refer to Section 5.9.3. **NOTE:** The Enumerated Types dictionaries populate the Version element using information from the **DT_Version** tag.
Type	DataTypes.UINT	Total range is from 0 to 255, where values 0 - 127 are reserved and values 28-255 are extensible. • Dictionary.Types.FIELD_DEFINITIONS == 1 • Dictionary.Types.ENUM_TABLES == 2 • Dictionary.Types.RECORD_TEMPLATES == 3 • Dictionary.Types.DISPLAY_TEMPLATES == 4 • Dictionary.Types.DATA_DEFINITIONS == 5 • Dictionary.Types.STYLE_SHEET == 6 • Dictionary.Types.REFERENCE == 7	**Required**. Indicates the type of dictionary contained in the payload.
DictionaryId	DataTypes.INT	Total range is from -16383 to 16383, where: • Values 0 to 16383 are reserved by Refinitiv • The value **1** corresponds to the **RDMFieldDictionary**. • Values -1 to -16383 are Extensible	An ID that can indicates a relationship between dictionaries. For example, a Field Dictionary and EnumeratedTypes Dictionary may have the same **DictionaryId**, indicating that they can be used together. **DictionaryId** defaults to **0**.
RT_Version	DataTypes.ASCII_STRING	"1.1"	Optionally sent only with the enumerated type dictionary. **RT_Version** identifies which field dictionary should be used with this enumerated type dictionary.
DT_Version	DataTypes.ASCII_STRING	"1.1"	Optionally sent only with the enumerated type dictionary. **DT_Version** conveys the display template version.

Table 39: Dictionary `summaryData`

5.5 Field Dictionary

5.5.1 Field Dictionary Payload

The payload of a Field Dictionary Refresh Message consists of a **`Series`** where each series entries contains a **`ElementList`**. Each **`SeriesEntry`** represents a row of information in the dictionary. The **`ElementList`** contained in each series entry provides information about an element of the row.

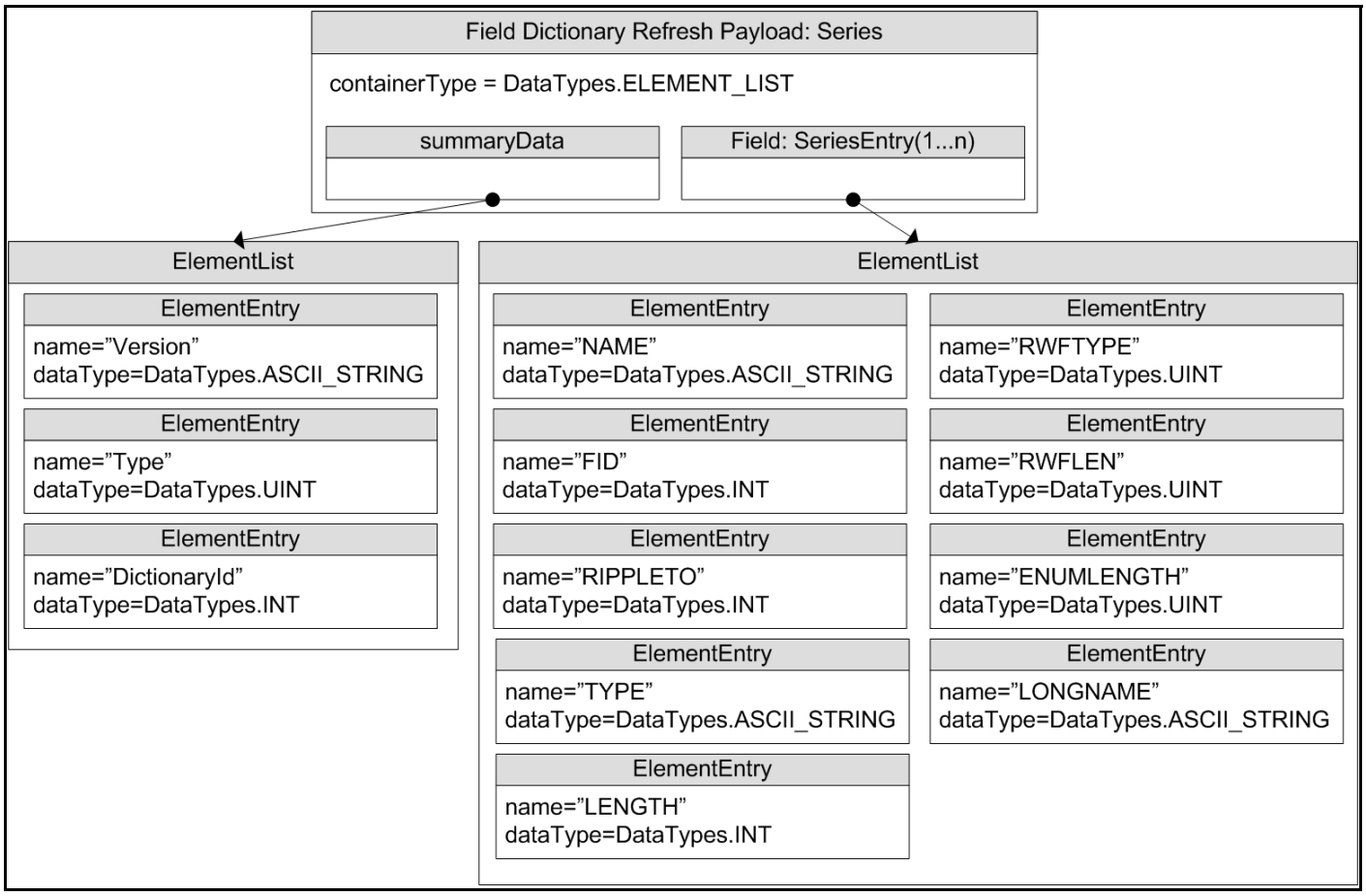


Figure 8. Field Dictionary Payload

Element entries do not have default values.

NAME	TYPE	LEAST VERBOSITY	RANGE/EXAMPLE	DESCRIPTION
NAME	DataTypes.ASCII_STRING	MINIMAL	e.g., “PROD_PERM”	Equivalent to the field’s **ACRONYM** (i.e., Short Name).
FID	DataTypes.INT	MINIMAL	-32768 to 32767	The field’s **`fieldId`**.
RIPPLETO	DataTypes.INT	MINIMAL	-32768 to 32767	If the field ripples, this is the **`fieldId`** of the field it ripples to. A value of **0** indicates no rippling. For a description of rippling, refer to the *Enterprise Transport API Java Edition Developers Guide*.
TYPE[a]	DataTypes.INT	MINIMAL	e.g., MfFieldTypes.INTEGER	The data type of the field for the Marketfeed format.
LENGTH[a]	DataTypes.UINT	MINIMAL	0 to 65535	The maximum string length of the field for the Marketfeed format.
RWFTYPE	DataTypes.UINT	MINIMAL	e.g., DataTypes.INT	The data type (**`DataTypes`**) of the field.
RWFLEN	DataTypes.UINT	MINIMAL	0 to 65535	The maximum length needed to cache the encoded value (the value found in the **`FieldEntry`**’s **`encData`** buffer). This is only a suggestion and is not enforced. A length of 0 implies that the maximum possible size for that type should be used for caching.
ENUMLENGTH	DataTypes.UINT	NORMAL	0 to 65535	Used for fields of type **DataTypes.ENUM**. This is the length of the **DISPLAY** element in its Enumerated Types table (See Section 5.6.1).
LONGNAME	DataTypes.ASCII_STRING	NORMAL	e.g., “PERMISSION”	Equivalent to the field’s **DDE ACRONYM** (i.e., Long Name).

Table 40: Field Dictionary Element Entries

a. These elements are specific to the Marketfeed format and can be used in converting to or from it. They can otherwise be ignored.

5.5.2 Field Dictionary File Format

The **RDMFieldDictionary** file format is a plain-text table. Each row represents one field, and each column a datum about that field. Each row is separated with a line break and columns are separated by whitespace. Lines beginning with an exclamation point (**!**) are comments and are ignored.

```
!ACRONYM     DDE ACRONYM        FID   RIPPLES TO   FIELD TYPE     LENGTH    RWF TYPE          RWF LEN
PROD_PERM    "PERMISSION"       1     NULL         INTEGER        5         UINT64            2
RDNDISPLAY   "DISPLAYTEMPLATE"  2     NULL         INTEGER        3         UINT32            1
DSPLY_NAME   "DISPLAY NAME"     3     NULL         ALPHANUMERIC   16        RMTES_STRING      16
RDN_EXCHID   "IDN EXCHANGE ID"  4     NULL         ENUMERATED     3 ( 3 )   ENUM              1
TRDPRC_2     "LAST 1"           7     TRDPRC_3     PRICE          17        REAL64            7
TRDPRC_3     "LAST 2"           8     TRDPRC_4     PRICE          17        REAL64            7
TRDPRC_4     "LAST 3"           9     TRDPRC_5     PRICE          17        REAL64            7
TRDPRC_5     "LAST 4"           10    NULL         PRICE          17        REAL64            7
```

Figure 9. Field Dictionary File Format Sample

Several tagged attributes are available at the beginning of the file. These attributes provide versioning information about the dictionary in the file and are processed when loading from a file-based dictionary. Some of this information is conveyed along with the domain model representation of the dictionary. Tags may be added as future dictionary versions become available.

For the **RDMFieldDictionary**, an example of these tags are shown below.

```
!tag Filename      RWF.DAT
!tag Desc          RDFD RWF field set
!tag Type          1
!tag Version       4.00.14
!tag Build         002
!tag Date          18-Nov-2010
```

Figure 10. Field Dictionary Tagged Attributes Sample

5.5.2.1 Field Dictionary Tag Attributes

The following table describes tag attributes and indicates whether they are used when encoding the domain representation of the file.

TAG ATTRIBUTE	DESCRIPTION
Filename	The original name of the file as created by Refinitiv. This typically will not match the current name of the file, **RDMFieldDictionary**. **Filename** is not used when encoding the domain representation of the field dictionary.
Desc	Describes the dictionary. **Desc** is not used when encoding the domain representation of the field dictionary.
Type	Stores the dictionary type associated with this dictionary. For a field dictionary, this should be **Dictionary.Types.FIELD_DEFINITIONS == 1**. Other types are defined in Section 5.4. **Type** is used when encoding the domain representation of the field dictionary.
Version	Stores version information associated with this dictionary. **Version** is used when encoding the domain representation of the field dictionary.
Build	Stores internal build information. **Build** is not used when encoding the domain representation of the field dictionary.
Date	Stores dictionary release date information. **Date** is not used when encoding the domain representation of the field dictionary.

Table 41: Field Dictionary File Tag Information

5.5.2.2 Field Dictionary Columns

The columns in the field dictionary correspond to the **`ElementEntry`** names used while encoding and decoding the Field Dictionary:

COLUMN NAME IN FILE	REFINITIV WIRE FORMAT ELEMENT NAME	NOTES
ACRONYM	NAME	The abbreviated name corresponding to the field.
DDE ACRONYM	LONGNAME	A longer version of the name represented by the Acronym.
FID	FID	The Field IDentifier value.
RIPPLES TO	RIPPLETO	The file format uses the **`ACRONYM`** of the target field, rather than the rows **`fieldId`**. If the field does not ripple, this should be **NULL**.
FIELD TYPE	TYPE	The Marketfeed type associated with this field.
LENGTH	LENGTH (ENUMLENGTH)	The Marketfeed length associated with the field.
RWF TYPE	RWFTYPE	The Refinitiv Wire Format type (**`DataTypes`**) associated with the field.
RWF LEN	RWFLEN	A caching length hint associated with this field.

Table 42: Field Dictionary File Column Names and `ElementEntry` Names

5.5.2.3 RWF TYPE Keywords

The following keywords are supported for the RWFTYPE:

KEYWORD	DATA TYPE
ANSI_PAGE[a]	DataTypes.ANSI_PAGE
ARRAY[a]	DataTypes.ARRAY
ASCII_STRING	DataTypes.ASCII_STRING
BUFFER	DataTypes.BUFFER
DATE	DataTypes.DATE
DATETIME[a]	DataTypes.DATETIME
DOUBLE[a]	DataTypes.DOUBLE
ELEMENT_LIST, ELEM_LIST[a]	DataTypes.ELEMENT_LIST
ENUM	DataTypes.ENUM
FIELD_LIST[a]	DataTypes.FIELD_LIST
FILTER_LIST[a]	DataTypes.FILTER_LIST
FLOAT[a]	DataTypes.FLOAT
INT, INT32, INT64	DataTypes.INT
MAP[a]	DataTypes.MAP
MSG[a]	DataTypes.MSG
OPAQUE	DataTypes.OPAQUE
QOS[a]	DataTypes.QOS
REAL, REAL32, REAL64	DataTypes.REAL
RMTES_STRING	DataTypes.RMTES_STRING
SERIES[a]	DataTypes.SERIES
STATUS[a]	DataTypes.STATE
TIME	DataTypes.TIME
UINT, UINT32, UINT64	DataTypes.UINT
UTF8_STRING	DataTypes.UTF8_STRING
VECTOR[a]	DataTypes.VECTOR
XML[a]	DataTypes.XML

Table 43: Field Dictionary Type Keywords and Associated Data Types

a. Type is Refinitiv Wire Format-Only and does not have a Marketfeed equivalent.

5.5.2.4 FIELD TYPE Keywords

Valid keywords for the Marketfeed Field Type are **INTEGER**, **ALPHANUMERIC**, **ENUMERATED**, **TIME**, **TIME_SECONDS**, **DATE**, or **PRICE**.

The table below lists the mappings from **FIELD TYPE** to the **RWFTYPE** keyword. All are used in **RDMFieldDictionary** and are safe.

FIELD TYPE	LENGTH	RWFTYPE	RWFLEN	NOTES
ALPHANUMERIC	14	ASCII_STRING	14	RIC/SYMBOL
ALPHANUMERIC	21	ASCII_STRING	21	RIC/SYMBOL
ALPHANUMERIC	28	ASCII_STRING	28	RIC/SYMBOL
ALPHANUMERIC	1-255	RMTES_STRING	1-255	length <= 3 is technically ASCII
ENUMERATED	2-3 (1-8)	ENUM	1	Enum values 0 - 255
ENUMERATED	5 (3-8)	ENUM	2	Enum values 0 - 65535
BINARY	3	UINT32	2	Base64 encoded 2-byte unsigned int
BINARY	4	UINT32	3	Base64 encoded 3-byte unsigned int
BINARY	43	BUFFER	32	Base64 encoded buffer
BINARY	171	BUFFER	128	Base64 encoded buffer
DATE	11	DATE	4	Day, month, year
TIME_SECONDS	8	TIME	5	Time in hour, minute, second, and millisecond
TIME	5	TIME	5	Time in hour, minute, and second
PRICE	17	REAL	9	**`Real`** can represent values with fractional denominators, trailing zeros, or up to 14 decimal positions.
INTEGER	15	REAL	7	Signed integer value, where trailing zero values can be optimized off of the wire.
INTEGER	3	UINT	1	unsigned int 0 - 255
INTEGER	5	UINT	2	unsigned int 0 - 65535
INTEGER	10	UINT	5	unsigned int 0 - (2^{40}-1)
INTEGER	15	UINT	8	unsigned int 0 - (2^{64}-1)
INTEGER	15	UINT	4	unsigned int 0 - (2^{32}-1)

Table 44: Marketfeed to Refinitiv Wire Format Mappings in RDMFieldDictionary

5.6 Enumerated Types Dictionary

5.6.1 Enumerated Types Dictionary Payload

The payload of an Enumerated Types Dictionary Refresh Message consists of a **`Series`** with each series entry (**`SeriesEntry`**) containing an **`ElementList`** and representing a table in the dictionary. The **`ElementList`** in each entry contains information about each Enumerated Type in the table.

Each **`ElementEntry`** has a type of **DataTypes.ARRAY**, where there is one element for each column in the file: **VALUE**, **DISPLAY**, and **MEANING**. The content of each **`Array`** corresponds to one Enumerated Type, so each array should contain the same number of entries.

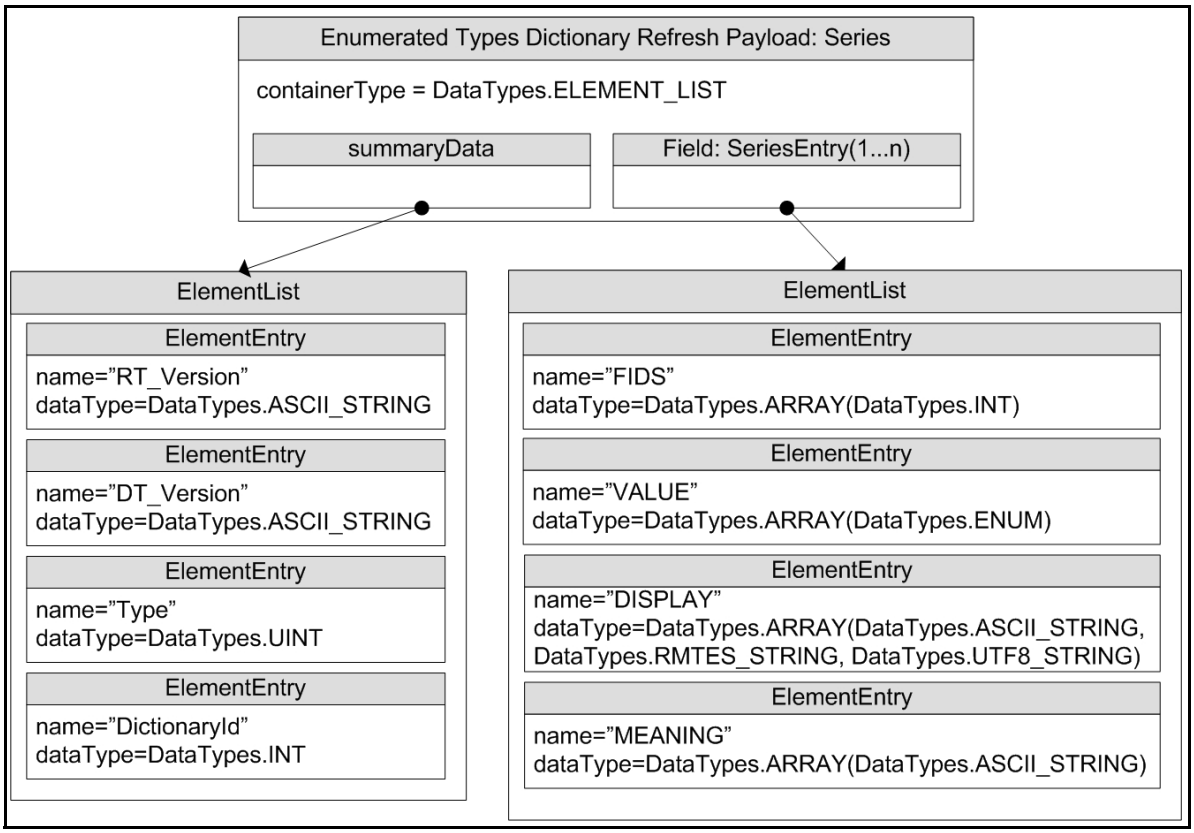


Figure 11. Enumerated Types Dictionary Refresh Message Payload

NAME	TYPE	LEAST VERBOSITY	EXAMPLE LIST	DESCRIPTION
FIDS	DataTypes.ARRAY of DataTypes.INT	NORMAL	15, 1084, 1085,…	The **fieldId**'s of all fields that reference this table. These fields should have type **DataTypes.ENUM** in the Field Dictionary and use the values given in the **VALUE** list. The **Array.itemLength** should be 2 because each **fieldId** is a two-byte, signed integer value.
VALUE	DataTypes.ARRAY of DataTypes.ENUM	NORMAL	826, 840, …	Includes values that correspond to each Enumerated Type. **FieldEntries** that use the table contain these values. The **Array.itemLength** should be 2 since each enum is a two-byte, unsigned integer value.
DISPLAY	DataTypes.ARRAY of DataTypes.ASCII_STRING, DataTypes.RMTES_STRING, or DataTypes.UTF8_STRING	NORMAL	"GBP", "USD",…	Brief, displayable names for each Enumerated Type. When special characters are needed, the DISPLAY column uses a hexadecimal value identified by using hash marks instead of quotation marks (e.g., **#42FE#**).
MEANING	DataTypes.ARRAY of DataTypes.ASCII_STRING	VERBOSE	"UK pound sterling", "US Dollar",…	A longer description of each Enumerated Type. **NOTE:** Providers do not need to provide this array (even when verbosity is **VERBOSE**).

Table 45: Element Entries Describing Each Enumerated Type Table

5.6.2 Enumerated Types Dictionary File Format

The **enumtype.def** file format is a plain-text set of tables. Rows are separated by lines and columns are separated by whitespace (excepting quoted strings, as illustrated in Section 5.6.1). Lines that begin with an exclamation point (**!**) are comments and are ignored.

The file contains a set of tables, each with two sections:

1. A section with the list of **fieldId** values corresponding to all fields that use the table.
2. A section with the table of enumerated values and their respective display data.

5.6.2.1 Enumerated Types Dictionary File Example

```
! ACRONYM    FID
! -------    ---
BIG_FIGURE   6207
PIPS_POS     6208
! VALUE      DISPLAY   MEANING
! -----      -------   -------
    0        "INT"     whole number
    1        "1DP"     1 decimal place
    2        "2DP"     2 decimal places
    3        "3DP"     3 decimal places
    4        "4DP"     4 decimal places
    5        "5DP"     5 decimal places
    6        "6DP"     6 decimal places
    7        "7DP"     7 decimal places
!
! ACRONYM     FID
! -------     ---
MATUR_UNIT    2378
!
! VALUE      DISPLAY   MEANING
! -----      -------   -------
    0        "   "   Undefined
    1        "Yr "   Years
    2        "Mth"   Months
    3        "Wk "   Weeks
    4        "Day"   Days
```

Code Example 5: Enumerated Types Dictionary File Format Sample

5.6.2.2 Tagged Attributes

Several tagged attributes are available at the beginning of the file. These attributes provide version information about the dictionary contained in the file and are processed while loading from a file-based dictionary. Some of this information is conveyed along with the domain model representation of the dictionary. Tags may be added as future dictionary versions become available.

For the **enumtype.def**, an example of these tags are as follows:

```
!tag Filename      ENUMTYPE.001
!tag Desc          IDN Marketstream enumerated tables
!tag Type          2
!tag RT_Version    4.20.17
!tag DT_Version    15.41
!tag Date          5-Feb-2017
```

Code Example 6: Enumerated Types Dictionary Tagged Attribute Sample

The following table describes the tag attributes and indicates which are used when encoding the domain representation of the file.

TAG ATTRIBUTE	DESCRIPTION
Date	Includes information regarding the dictionary release date. **Date** is not used when encoding the domain representation of the field dictionary.
Desc	A Description of the dictionary. **Desc** is not used when encoding the domain representation of the field dictionary.
DT_Version	The version of the display template version. **DT_Version** is used when encoding the domain representation of the field dictionary. For device compatibility purposes, this value is sent as both Version and DT_Version.
Filename	The original name of the file as created by Refinitiv. This typically does not match the current name of the file, **enumtype.def**. **Filename** not used when encoding the domain representation of the field dictionary.
RT_Version	The version of the field dictionary associated with this enumerated type dictionary. **RT_Version** is used when encoding the domain representation of the field dictionary.
Type	The dictionary type associated with this dictionary. For an enumerated types dictionary, this should be **Dictionary.Types.ENUM_TABLES == 2**. Other types are defined in Section 5.4. **Type** is used when encoding the domain representation of the field dictionary.

Table 46: Enumerated Type Dictionary File Tag Information

5.6.2.3 Reference Fields Section

The first section lists all fields that use the table. These fields should have the type **DataTypes.ENUM** in their corresponding Field Dictionary and have matching names.

NAME	REFINITIV WIRE FORMAT ELEMENT NAME
ACRONYM	n/a (The name of the field is not sent with the dictionary payload).
FID	FIDS

Table 47: Refinitiv Wire Format EnumType Dictionary File Format Reference Fields

5.6.2.4 Values Table Section

The second section lists the value of each enumerated type and its corresponding display data.

NAME	REFINITIV WIRE FORMAT ELEMENT NAME	NOTES
VALUE	VALUE	The unsigned, integer value corresponding to the enumerated value.
DISPLAY	DISPLAY	In cases where special characters are needed, the DISPLAY column uses a hexadecimal value, which is identified by using hash marks instead of quotation marks, e.g. **#42FE#**.
MEANING	MEANING	The meaning column is not required over the network and typically not provided.

Table 48: Refinitiv Wire Format EnumType Dictionary File Values

5.7 Global Field Set Definition Payload

NOTE: This portion of the Dictionary Domain is used only with the Elektron Direct Feed (EDF) product. For further details, consult the EDF product documentation.

The payload of a Field Set Definition Dictionary Refresh Message consists of a **`Vector`**, where each vector entry contains an **`ElementList`**. Each **`VectorEntry`** represents a specific Set Definition Id, which corresponds to the **`VectorEntry`**'s index. Each line in the set definition is represented by a corresponding value in the arrays within the FIDS and TYPES element entries.

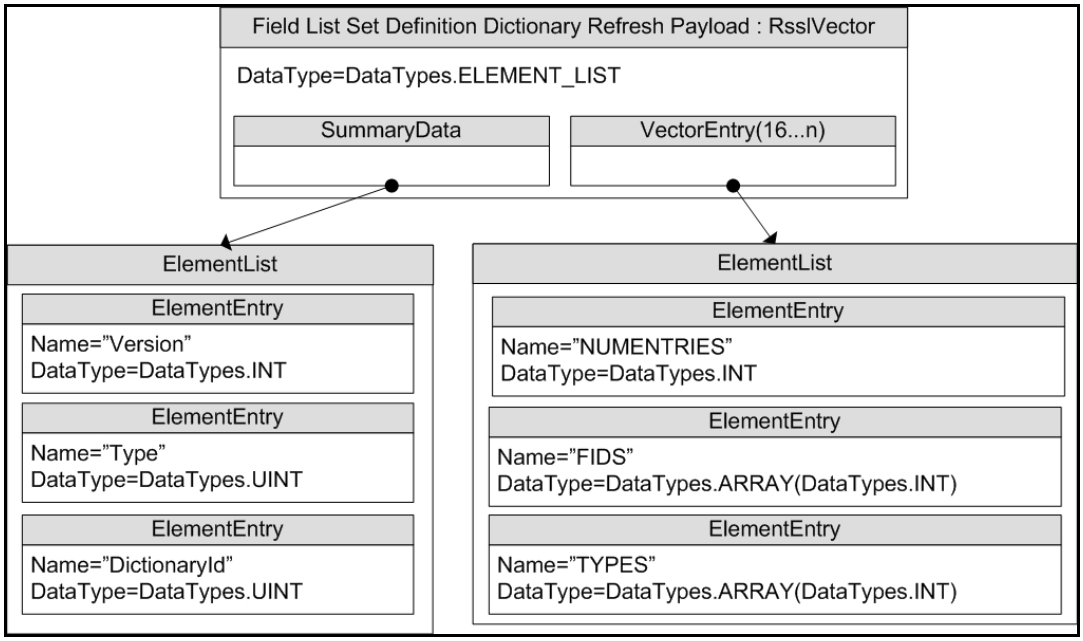


Figure 12. Global Field Set Definition Payload

The following table shows the valid type enumerations for a set definition line, including base primitives, as well as set primitives.

RWFTYPE	DESCRIPTION
DataTypes.INT	Signed Integer, 63 bits of value, with one bit of sign
DataTypes.UINT	Unsigned Integer, 64 bits of value
DataTypes.FLOAT	32-bit Floating Point, represents the same range as the system type
DataTypes.DOUBLE	64-bit Floating Point, represents the same range as the system type
DataTypes.REAL	Real Numeric value, 63 bits of value, with a one-bit sign, and a hint enumeration
DataTypes.DATE	Date type, containing month, day, and year values
DataTypes.TIME	Time type, containing hour, minute, second, and millisecond values
DataTypes.DATETIME	Date time type, containing all members of **`DataTypes.DATE`** and **`DataTypes.TIME`**
DataTypes.QOS	Quality of Service type
DataTypes.STATE	Stream state type.
DataTypes.ENUM	Enumeration type, defined as a two-byte unsigned value
DataTypes.ARRAY	Array type, represents a list of simple primitive types
DataTypes.BUFFER	Buffer type, represents a binary data buffer
DataTypes.ASCII_STRING	Buffer type containing ASCII string data
DataTypes.UTF8_STRING	Buffer type containing UTF8 string data
DataTypes.RMTES_STRING	Buffer type containing RMTES string data
DataTypes.INT_1	1-byte encoded length signed integer, 7 bits value with 1 bit sign
DataTypes.INT_2	2-byte encoded length signed integer, 15 bits value with 1 bit sign
DataTypes.INT_4	4-byte encoded length signed integer, 31 bits value with 1 bit sign
DataTypes.INT_8	8-byte encoded length signed integer, 63 bits value with 1 bit sign
DataTypes.UINT_1	1 encoded byte length unsigned integer, 8 bits of value
DataTypes.UINT_2	2-byte encoded length unsigned integer, 16 bits of value
DataTypes.UINT_4	4-byte encoded length unsigned integer, 32 bits of value
DataTypes.UINT_8	8-byte encoded length unsigned integer, 64 bits of value
DataTypes.FLOAT_4	4-byte encoded length floating point value
DataTypes.DOUBLE_8	8-byte encoded length floating point value
DataTypes.REAL_4RB	Optimized, variable length real encoding, with a maximum value up to 31 bits and 1 signed bit
DataTypes.REAL_8RB	Optimized, variable length real encoding, with a maximum value of 63 bits and 1 signed bit
DataTypes.DATE_4	4-byte encoded length date type containing year, month, and day values
DataTypes.TIME_3	3-byte encoded length time type containing hours, minutes, and seconds values
DataTypes.TIME_5	5-byte encoded length time type containing hours, minutes, seconds, and milliseconds values

Table 49: Set Definition Enumerations

RWFTYPE	DESCRIPTION
DataTypes.TIME_7	7-byte encoded length time type containing hours, minutes, seconds, milliseconds, and microseconds values
DataTypes.TIME_8	8-byte encoded length time type containing hours, minutes, seconds, milliseconds, microseconds, and nanoseconds values
DataTypes.DATETIME_7	7-byte encoded length date time type containing full **`DataTypes.DATE`** and hours, minutes and seconds
DataTypes.DATETIME_9	9-byte encoded length date time type containing full **`DataTypes.DATE`** and **`DataTypes.TIME`** values
DataTypes.DATETIME_11	11-byte encoded length date time type containing DataTypes.DATE and hours, minutes, seconds, milliseconds, and microseconds values.
DataTypes.DATETIME_12	12-byte encoded length date time type containing RSSL_DT_DATE/DataTypes.DATE and hours, minutes, seconds, milliseconds, microseconds, and nanoseconds values

Table 49: Set Definition Enumerations (Continued)

5.8 Global Element Set Definition Payload

NOTE: This portion of the Dictionary Domain is used only with the Elektron Direct Feed (EDF) product. For further details, consult the EDF product documentation.

The payload of a Global Element Set Definition Dictionary Refresh Message consists of a **`Vector`**, where each vector entry contains an **`ElementList`**. Each **`VectorEntry`** represents a specific set definition ID, which corresponds to the **`VectorEntry`**'s index. Each line in the set definition is represented by a corresponding value in the arrays within the FIDS and TYPES element entries.

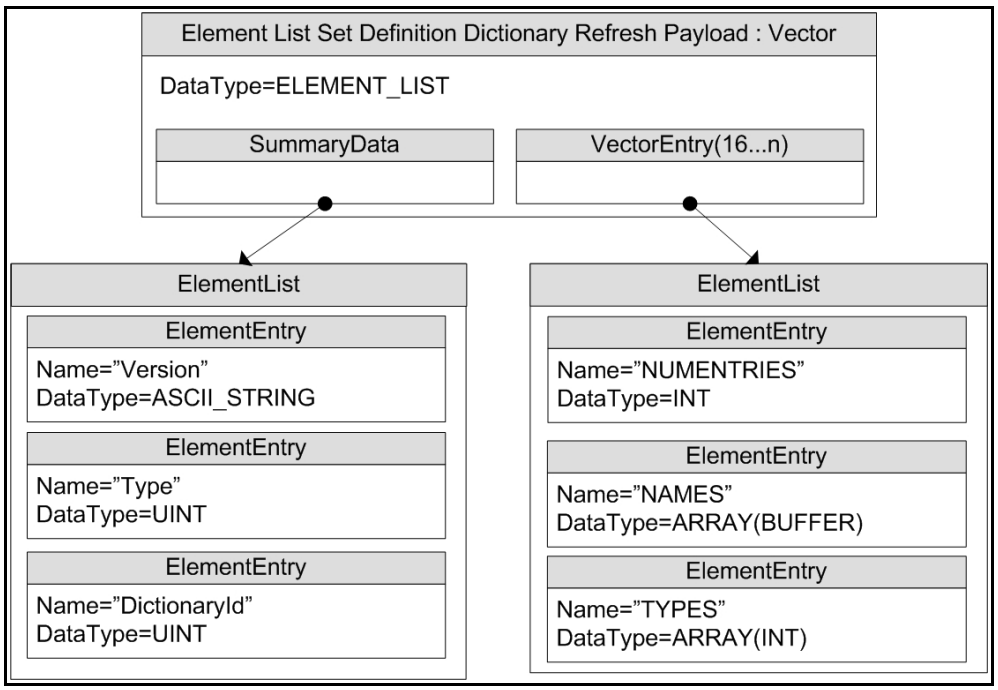


Figure 13. Global Element Set Definition Payload

5.9 Special Semantics

5.9.1 DictionaryId

A **dictionaryId** can be included with **FieldList** and **ElementList** payload information. The **dictionaryId** contained in the payload should match the Id of the dictionaries required to encode or decode the content. (e.g. Field Dictionary, Enumerated Types Dictionary). Dictionaries used together to decode a field list should all have a matching Id.

dictionaryId values are globally scoped and can have a range of values from -16383 to 16383. Values 0 through 16383 are reserved for Refinitiv's use. Customers can provide their own dictionaries by selecting a **dictionaryId** between -1 and -16383.

A **dictionaryId** can be changed in two ways. If a **dictionaryId** is provided on a solicited or unsolicited refresh message, this dictionary is used across all messages on the stream until a different **dictionaryId** is provided in a subsequent solicited or unsolicited refresh. If a **FieldEntry** contains a **fieldId** of zero, this reserved value indicates a temporary dictionary change. In this situation, this entry's value is the new dictionaryId, encoded or decoded as a **Int**. When a **dictionaryId** is changed in this manner, the change is only in effect on the remaining entries in the field list or until another **fieldId** of zero is encountered. Any other **containerType** housed inside of a remaining entry also adopts the temporarily changed dictionary. After the end of the field list is reached, the dictionaryId from the refresh takes precedence once again.

5.9.2 DictionariesProvided and DictionariesUsed

The Source Directory's Info filter entry (refer to Section 4.3.1.1) includes two elements related to Dictionaries: **DictionariesProvided** and **DictionariesUsed**. Both elements contain an **Array** of **DataTypes.ASCII_STRING** dictionary names. These names can be used in **msgKey.name** to request the dictionaries.

▶ To dynamically discover dictionaries while minimizing the amount of data downloaded:

1. Parse the "DictionariesUsed" from each desired service in the Source Directory.
2. Parse the "DictionariesProvided" from every service in the Source Directory.
3. Make a streaming request for each Dictionary Name listed in **DictionariesUsed** from the service that lists the **DictionaryName** in **DictionariesProvided**. The **msgKey.filter** of each **RequestMsg** should be set to **Dictionary.VerbosityValues.INFO** so as to request only the dictionary's basic information (because dictionaries tend to be large, this setting prevents unnecessary network traffic).
4. For each Dictionary response, parse the **summaryData** in the payload to obtain the dictionary's Type and Version.
   - If a dictionary is of an unneeded type, that dictionary stream can be closed.
   - If a dictionary is needed, a reissue request can be made where the **msgKey.filter** requests a higher verbosity (e.g. **Dictionary.VerbosityValues.NORMAL**).
   - Version information can be used to determine if the consumer needs to update its dictionary.

5.9.3 Version Information

The version of a dictionary is normally available in Summary Data in the payload of a **RefreshMsg**. All available verbosities are expected to include this information. The verbosity **Dictionary.VerbosityValues.INFO** can be used to request only the version information (as the many fields in dictionaries tend to result in large messages).

This information normally comes in the form of a **DataTypes.ASCII_STRING** containing a dotted-decimal version number, indicating first the major version, followed by the minor version, and possibly followed by a third (informational) micro-version. For example, in the string **1.2.3**:

- **1** is the major version
- **2** is the minor version
- **3** is the micro-version

5.9.3.1 Version Information Usage

Version information has a couple of uses:

- The ***minor*** version changes whenever a dictionary adds new fields, but does not modify existing fields. This means the consumer can still use the previous dictionary with its data (though the consumer is unable to decode any new fields). Also, if the consumer has multiple dictionaries with the same major version available, it can use the minor version information to determine which is the latest (and therefore will be able to decode all fields regardless of the data's source).
- The ***major*** version changes if the dictionary changes in a way that is not compatible with previous versions (such as changing an existing field). This means that data encoded using a dictionary with one major version cannot be decoded using a dictionary with a different major version. If a consumer learns that its provider has changed to a dictionary with a different major version, it must retrieve the new dictionary before again decoding data.

5.9.3.2 Handling Dictionary Version Changes

To keep consumers informed of changes, Refinitiv recommends that dictionary requests be streaming even though updates are not used for this domain.

If the dictionary's minor version changes, a provider may advertise it via a **`StatusMsg`** with a **`State`** of **StreamStates.OPEN**/ **DataStates.SUSPECT**. The consumer may then reissue its dictionary request to obtain the latest version.

If a dictionary's major version is changed, the provider should disconnect all consumers to ensure that the consumers' content and dictionary are entirely resynchronized.

5.10 Other Dictionary Types

The Dictionary domain is intended to be used for other versionable data that updates very rarely. This section briefly describes the other reserved dictionary types.

None of these dictionary types are currently used, nor is there any domain model specification associated with any of them at this time.

DICTIONARY TYPE	DESCRIPTION
RecordTemplate	A ***RecordTemplate*** dictionary contains multiple record templates. Each record template contains a list of all of the fields present in a FieldList. Historically, RecordTemplates have been used for QForms. While they can be used for Marketfeed records and FieldLists, they typically are not used for those data formats. Templates for Marketfeed records and FieldLists are dynamically generated when an image is received. Examples of external file representations for a RecordTemplate dictionary include **tss_records.cf** and **appendix_d**.
DisplayTemplate	A ***DisplayTemplate*** dictionary contains specifications that describe how and where to display fields on a screen.
DataDefinition	A Data Definitions dictionary contains specifications for Set Definitions for use with Field Lists and Element Lists. Set Definitions provide bandwidth optimization by separating out repetitive or redundant data.
StyleSheet	A ***StyleSheet*** dictionary contains an XSLT or CSS style sheet.
Reference	A ***Reference*** dictionary is a table of reference information provided as a Series. This information is not used for parsing, interpreting, caching, or displaying data.

Table 50: Other Dictionary Types

5.11 Dictionary Sample XML

5.11.1 Dictionary Request Message Sample XML

```
<REQUEST domainType="DICTIONARY" streamId="3" containerType="NO_DATA" flags="0x00" dataSize="0">
   <key flags="0x0B (HAS_SERVICE_ID|HAS_NAME|HAS_FILTER)" serviceId="1257" name="RWFFld" filter="7"/>
   <dataBody>
   </dataBody>
</REQUEST>
```

Code Example 7: Dictionary Request Message Sample XML Message Layout

5.11.2 Field Dictionary Refresh Message Sample XML

```
<REFRESH domainType="DICTIONARY" streamId="3" containerType="SERIES" flags="0x168
        (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|CLEAR_CACHE)" groupId="0" State: Non-streaming/Ok/None
        - text: "" dataSize="362816">
    <key flags="0x0B (HAS_SERVICE_ID|HAS_NAME|HAS_FILTER)" serviceId="1257 name="RWFFld" filter="7"/>
    <dataBody>
        <series  flags="0x07 (HAS_SET_DEFS|HAS_SUMMARY_DATA|HAS_TOTAL_COUNT_HINT)" countHint="8984"
                containerType="ELEMENT_LIST">
            <elementSetDefs>
                <elementSetDef setId="0">
                    <elementSetDefEntry name="NAME" dataType="ASCII_STRING" />
                    <elementSetDefEntry name="FID" dataType="INT_2" />
                    <elementSetDefEntry name="RIPPLETO" dataType="INT_2" />
                    <elementSetDefEntry name="TYPE" dataType="INT_1" />
                    <elementSetDefEntry name="LENGTH" dataType="UINT_2" />
                    <elementSetDefEntry name="RWFTYPE" dataType="UINT_1" />
                    <elementSetDefEntry name="RWFLEN" dataType="UINT_2" />
                    <elementSetDefEntry name="ENUMLENGTH" dataType="UINT_2" />
                    <elementSetDefEntry name="LONGNAME" dataType="ASCII_STRING" />
                </elementSetDef>
            </elementSetDefs>
            <summaryData>
                <elementList flags="0x08 (HAS_STANDARD_DATA)">
                    <elementEntry name="Type" dataType="UINT" data="1"/>
                    <elementEntry name="DictionaryId" dataType="INT" data="1"/>
                    <elementEntry name="Version" dataType="ASCII_STRING" data="4.10.15"/>
                </elementList>
            </summaryData>
            <seriesEntry>
                <elementList flags="0x02 (HAS_SET_DATA)">
                    <elementEntry name="NAME" dataType="ASCII_STRING" data="PROD_PERM"/>
                    <elementEntry name="FID" dataType="INT" data="1"/>
                    <elementEntry name="RIPPLETO" dataType="INT" data="0"/>
                    <elementEntry name="TYPE" dataType="INT" data="1"/>
```

```
                    <elementEntry name="LENGTH" dataType="UINT" data="5"/>
                    <elementEntry name="RWFTYPE" dataType="UINT" data="4"/>
                    <elementEntry name="RWFLEN" dataType="UINT" data="2"/>
                    <elementEntry name="ENUMLENGTH" dataType="UINT" data="0"/>
                    <elementEntry name="LONGNAME" dataType="ASCII_STRING" data="PERMISSION"/>
                </elementList>
            </seriesEntry>
            <seriesEntry>
                <elementList flags="0x02 (HAS_SET_DATA)">
                    <elementEntry name="NAME" dataType="ASCII_STRING" data="RDNDISPLAY"/>
                    <elementEntry name="FID" dataType="INT" data="2"/>
                    <elementEntry name="RIPPLETO" dataType="INT" data="0"/>
                    <elementEntry name="TYPE" dataType="INT" data="1"/>
                    <elementEntry name="LENGTH" dataType="UINT" data="3"/>
                    <elementEntry name="RWFTYPE" dataType="UINT" data="4"/>
                    <elementEntry name="RWFLEN" dataType="UINT" data="1"/>
                    <elementEntry name="ENUMLENGTH" dataType="UINT" data="0"/>
                    <elementEntry name="LONGNAME" dataType="ASCII_STRING" data="DISPLAYTEMPLATE"/>
                </elementList>
            </seriesEntry>
            <!-- Additional entries... -->
        </series>
    </dataBody>
</REFRESH>
```

Code Example 8: Field Dictionary Refresh Message Sample XML Message Layout

5.11.3 Enumerated Types Dictionary Refresh Message Sample XML

```
<REFRESH domainType="DICTIONARY" streamId="4" containerType="SERIES" flags="0x168
        (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|CLEAR_CACHE)" groupId="0" State: Non-streaming/Ok/None
        - text: "" dataSize="60161">
    <key flags="0x0B (HAS_SERVICE_ID|HAS_NAME|HAS_FILTER)" serviceId="1257" name="RWFEnum" filter="7"/
            >
    <dataBody>
        <series  flags="0x03 (HAS_SET_DEFS|HAS_SUMMARY_DATA)" countHint="0"
                containerType="ELEMENT_LIST">
            <elementSetDefs>
                <elementSetDef setId="0">
                    <elementSetDefEntry name="FIDS" dataType="ARRAY" />
                    <elementSetDefEntry name="VALUE" dataType="ARRAY" />
                    <elementSetDefEntry name="DISPLAY" dataType="ARRAY" />
                </elementSetDef>
            </elementSetDefs>
            <summaryData>
                <elementList flags="0x08 (HAS_STANDARD_DATA)">
                    <elementEntry name="Type" dataType="UINT" data="2"/>
                    <elementEntry name="DictionaryId" dataType="INT" data="1"/>
```

```
                    <elementEntry name="Version" dataType="ASCII_STRING" data="1.1"/>
                </elementList>
            </summaryData>
            <seriesEntry>
                <elementList flags="0x02 (HAS_SET_DATA)">
                    <elementEntry name="FIDS" dataType="ARRAY">
                        <array itemLength="2" primitiveType="INT">
                            <arrayEntry data="4"/>
                        </array>
                    </elementEntry>
                    <elementEntry name="VALUE" dataType="ARRAY">
                        <array itemLength="2" primitiveType="ENUM">
                            <arrayEntry data="0"/>
                            <arrayEntry data="1"/>
                            <arrayEntry data="2"/>
                            <arrayEntry data="3"/>
                            <arrayEntry data="4"/>
                            <arrayEntry data="5"/>
                            <!-- Additional entries... -->
                        </array>
                    </elementEntry>
                    <elementEntry name="DISPLAY" dataType="ARRAY">
                        <array itemLength="3" primitiveType="ASCII_STRING">
                            <arrayEntry data="   "/>
                            <arrayEntry data="ASE"/>
                            <arrayEntry data="NYS"/>
                            <arrayEntry data="BOS"/>
                            <arrayEntry data="CIN"/>
                            <arrayEntry data="PSE"/>
                            <!-- Additional entries... -->
                        </array>
                    </elementEntry>
                </elementList>
            </seriesEntry>
            <!-- Additional entries... -->
        </series>
    </dataBody>
</REFRESH>
```

Figure 14. Enumerated Type Dictionary Refresh Message Sample XML Message Layout

5.12 Dictionary Utility Functions

Utility functions load, encode, and decode Field and Enumerated Types Dictionaries. You may find these functions in the **DataDictionary** class.

The functions should be called with a **`DataDictionary`** object, which stores information associated with each field (any associated Enumerated Types table, if the field uses one). Each load or decode function adds information to the object when called. Fields and Enumerated Types tables may be added from any number of sources, as long as there is no duplicate or conflicting information.

The **`DataDictionary`** object stores an array of **`DictionaryEntry`** objects, indexed by **`fieldId`**. Each entry stores the data type (**`DataTypes`**) of the field along with other information. The field will also have a reference to a **`EnumTypeTable`**, if the field uses an Enumerated Types table.

FUNCTION NAME	DESCRIPTION
DataDictionary.clear	Clears the **`DataDictionary`**. This must be done prior to initial use of the object.
DataDictionary.loadFieldDictionary	Adds data from a Field Dictionary file to the **`DataDictionary`** (refer to Section 5.5.2). This currently supports the **RDMFieldDictionary**, or other dictionaries using the same formatting.
DataDictionary.decodeFieldDictionary	Adds data from a Field Dictionary message payload to the **`DataDictionary`** (refer to Section 5.5.1).
DataDictionary.encodeFieldDictionary	Creates a Field Dictionary message payload from the **`DataDictionary`** object (refer to Section 5.5.1).
DataDictionary.loadEnumTypeDictionary	Adds data from an Enumerated Types Dictionary file to the **`DataDictionary`** (refer to Section 5.6.2). This currently supports the enumtype.def, or other enumerated type dictionaries using the same formatting.
DataDictionary.decodeEnumTypeDictionary	Adds data from an Enumerated Types Dictionary message payload to the **`DataDictionary`** (refer to Section 5.6.1).
DataDictionary.encodeEnumTypeDictionary	Creates an Enumerated Types Dictionary message payload from the **`DataDictionary`** object (refer to Section 5.6.1). Note that this function uses **DataTypes.ASCII_STRING** as the **DISPLAY** array type.
DataDictionary.toString()	Prints the information (fields, enumerated type tables) currently stored in the **`DataDictionary`**. Aids in debugging.
DataDictionary.extractDictionaryType	Extracts the Refinitiv Domain Model Dictionary Type (**`Dictionary.Types`**) information from an encoded dictionary. This can determine the specific dictionary decode function that should be used (e.g. for dictionary type of **Dictionary.Types.FIELD_DEFINITIONS**, the user would next invoke the **`DataDictionary.decodeFieldDictionary`** function). This is expected to be called after **`Msg.decode`** (where the **`Msg.domainType`** is **DomainTypes.DICTIONARY**), but before decoding the **`Msg.encodedDataBody`** payload.
FieldSetDefDb.clear	Clears the **`FieldSetDefDb`**. This must be called before initial use of the object.
FieldSetDefDb.addSetDef	Deep copies the given **`FieldSetDef`** into the **`FieldSetDefDb`**.
FieldSetDefDb.encode	Creates a Global Field Set Definition Dictionary message payload from the **`FieldSetDefDb`** object (refer to Section 5.7).
FieldSetDefDb.decode	Adds data from a Global Field Set Definition Dictionary message payload to the **`FieldSetDefDb`** object.
ElementSetDefDb.clear	Clears the **`ElementSetDefDb`**. This must be called before initial use of the object.
ElementSetDefDb.addSetDef	Deep copies the given **`ElementSetDef`** into the **`ElementSetDefDb`**.

Table 51: Dictionary Helper Functions

FUNCTION NAME	DESCRIPTION
ElementSetDefDb.encode	Creates a Global Element Set Definition Dictionary message payload from the **`ElementSetDefDb`** object (refer to Section 5.7).
ElementSetDefDb.decode	Adds data from a Global Element Set Definition Dictionary message payload to the **`ElementSetDefDb`** object.

Table 51: Dictionary Helper Functions (Continued)

5.12.1 Example: Basic Dictionary Use

The following example illustrates basic field and enumerated dictionary loading and unloading. Additional examples of encoding or decoding from domain model can be seen in the example applications provided with the Enterprise Transport API.

```
/* This sample shows loading field and enumerated dictionaries from a file */
DataDictionary dictionary = CodecFactory.createDataDictionary();

/* clear the dictionary before first use/load */
dictionary.clear();

/* load field dictionary from file */
if (dictionary.loadFieldDictionary(FIELD_DICTIONARY_FILE_NAME, error) < CodecReturnCodes.SUCCESS)
{
    System.out.println("Unable to load field dictionary.\n\tText: " + error.text());
    /* cleanup and exit */
}

/* load enumerated dictionary from file */
if (dictionary.loadEnumTypeDictionary(ENUM_TABLE_FILE_NAME, error) < CodecReturnCodes.SUCCESS)
{
    System.out.println("Unable to load enum dictionary.\n\tText: " + error.text());
    /* cleanup and exit */
}

/* when users are done, they should unload dictionaries to clean up memory */
dictionary.clear();
```

Code Example 9: Enterprise Transport API Dictionary Loading Utility Function Use

5.12.2 Example: Dictionary Lookup Using a FieldId

This example illustrates a dictionary lookup using a **fieldId** to determine how to decode the **FieldEntry** data. This example also cross references an enumerated type (**DataTypes.ENUM**) to use enumerated type dictionary information. For additional examples of encoding or decoding from the domain model, refer to the Enterprise Transport API's example applications.

```
/* This sample shows use of a loaded dictionary. This looks up a fieldId to determine how to decode the
        FieldEntry content. It also demonstrates cross referencing an enumerated type */

/* (DataDictionary) dictionary loaded earlier */

/* used while decoding field list content */
if ((ret = fieldEntry.decode(dIter)) != CodecReturnCodes.SUCCESS)
{
    System.out.println("decoding of FieldEntry failed with return code: " + ret);
    return;
}

DictionaryEntry dictionaryEntry = dictionary.entry(fieldEntry.fieldId());
if (dictionaryEntry == null)
{
    System.out.println(FieldId " + fieldEntry.fieldId() + " not found in dictionary");
    return;
}

/* If found, we can access various pieces of information, and use the Type information for further
        decoding of the field */

switch (dataType)
{
    case DataTypes.UINT:
        decodeUInt(dIter, uintValue);
        break;
    case DataTypes.INT:
        decodeInt(dIter, intValue);
        break;
    case DataTypes.ENUM:
        /* enumerated types can be cross referenced with enum dictionary */
        decodeEnum(dIter, enumValue);
        /* EnumType contains value, display, and meaning information */
        EnumType enumType = dictionary.entryEnumType(dictionaryEntry, enumValue);
        break;
        /* continued with decoding for other possible types */
}
```

Code Example 10: Enterprise Transport API Dictionary Utility Function Field and Enum Look Up Example

6 Market Price Domain

6.1 Description

The ***Market Price*** domain provides access to Level I market information such as trades, indicative quotes, and top-of-book quotes. All information is sent as a **`FieldList`**. Field-value pairs contained in the field list include information related to that item (i.e., net change, bid, ask, volume, high, low, or last price).

NOTE: **`GenericMsg`**(s) are not supported in the **MARKET_PRICE** Refinitiv Domain Model.

6.2 Usage

6.2.1 Market Price Request Message

A Market Price request message is encoded and sent by Open Message Model consumer applications. The request specifies the name and attributes of an item in which the consumer is interested.

To receive updates, a consumer can make a “streaming” request by setting the **RequestMsgFlags.STREAMING** flag. If the flag is not set, the consumer requests a “snapshot,” and the refresh ends the request.

To stop updates, a consumer can pause an item (if the provider supports the pause feature). For additional details, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.MARKET_PRICE == 6**
qos	Optional. Indicates the QoS at which the consumer wants the stream serviced. If both **`qos`** and **`worstQos`** are specified, this request can be satisfied by a range of QoS.
worstQos	Optional. Used with the **`qos`** member to define a range of acceptable QoS. When the provider encounters such a range, it should attempt to provide the best QoS it can within that range. **`worstQos`** should only be used on services that claim to support it via the **SupportsQosRange** item in the Source Directory response (refer to Section 4.3.1.1).
priorityClass	Optional. Indicates the class of a streams priority.
priorityCount	Optional. Indicates the count associated with a streams priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service from which the consumer wishes to request the item.
msgKey.nameType	Optional. When consuming from Refinitiv sources, typically set to **InstrumentNameTypes.RIC** (the “Reuters Instrument Code”). If unspecified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request, otherwise optional**. Specifies the name of the requested item.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.

Table 52: Market Price Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
Payload	Optional. When features such as View (**RequestMsgFlags.HAS_VIEW**) or Batch (**RequestMsgFlags.HAS_BATCH**) are leveraged, the payload can contain information relevant to that feature. For more detailed information, refer to the *Enterprise Transport API Java Edition Developers Guide*.

Table 52: Market Price Request Message Member Use (Continued)

6.2.2 Market Price Refresh Message

A Market Price Refresh Message is sent by Open Message Model provider and non-interactive provider applications. This message sends all currently available information about the item to the consumer.

`FieldList` in the payload should include all fields that may be present in subsequent updates, even if those fields are currently blank. When responding to a View request, this refresh should contain all fields that were requested by the specified view. If for any reason the provider wishes to send new fields, it must first send an unsolicited refresh with both the new and currently-present fields.

NOTE: All solicited or unsolicited refresh messages in the Market Price domain must be atomic, and have their **`RefreshMsgFlags.CLEAR_CACHE`** and **`RefreshMsgFlags.REFRESH_COMPLETE`** flags. The Market Price domain does not allow for multi-part refresh use.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REFRESH == 2**
partNum	Not used.
domainType	**Required**. **DomainTypes.MARKET_PRICE == 6**
state	**Required**. Includes the state of the stream and data.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies the permission information associated with content on this stream.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`msgKey.nameType`** should match the **`msgKey.nameType`** specified in the request. If unspecified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified. This should match the requested name.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. This should consist of a **`FieldList`** containing all fields associated with the item.

Table 53: Market Price Refresh Message Member Use

6.2.3 Market Price Update Message

A Market Price Update Message is sent by Open Message Model provider and non-interactive provider applications. The Market Price Update Message conveys any changes to an item's data.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.MARKET_PRICE == 6**
updateType	**Required**. Indicates the general content of the update: • **UpdateEventTypes.UNSPECIFIED == 0** • **UpdateEventTypes.QUOTE == 1** • **UpdateEventTypes.TRADE == 2** • **UpdateEventTypes.NEWS_ALERT == 3** • **UpdateEventTypes.VOLUME_ALERT == 4** • **UpdateEventTypes.ORDER_INDICATION == 5** • **UpdateEventTypes.CLOSING_RUN == 6** • **UpdateEventTypes.CORRECTION == 7** • **UpdateEventTypes.MARKET_DIGEST == 8** • **UpdateEventTypes.QUOTES_TRADE == 9** • **UpdateEventTypes.MULTIPLE == 10** • **UpdateEventTypes.VERIFY == 11**
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
conflationCount	Optional. If a provider sends a conflated update, **`conflationCount`** specifies the number of updates in the conflation. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
conflationTime	Optional. If a provider sends a conflated update, **`conflationTime`** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
permData	Optional. Specifies permissioning information associated with only the contents of this update.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the ID of the service that provides the data.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** should match the name type specified on the request. If **`msgKey.nameType`** is unspecified, its value defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **`msgKey.name`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.

Table 54: Market Price Update Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. This should consist of a `FieldList` with any changed data.

Table 54: Market Price Update Message Member Use (Continued)

6.2.4 Market Price Status Message

A Market Price Status Message is sent by Open Message Model provider and non-interactive provider applications. The status message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.MARKET_PRICE == 6**
State	Optional. Specifies the current state information associated with the data and stream.
groupId	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies permissioning information associated with only the contents of this message.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the ID of the service that provides the data.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** should match the name type specified on the request. If **`msgKey.nameType`** is unspecified, its value defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **`msgKey.name`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 55: Market Price Status Message Member Use

6.2.5 Market Price Post Message

If support is specified by the provider, consumer applications can post Market Price data. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

6.3 Data: Market Price Refresh Message / Update Message Payload

Market Price data is conveyed as an **`FieldList`**, where each **`FieldEntry`** corresponds to a piece of information and its current value. The field list should be decoded using its associated Field Dictionary, indicated by the **`dictionaryId`** present in the field list. For more information, refer to Section 5.2. For more information on using the **`FieldList`** container type, refer to the *Enterprise Transport API Java Edition Developers Guide*.

6.4 Market Price Sample XML

6.4.1 Market Price Request Message Sample XML

```
<REQUEST domainType="MARKET_PRICE" streamId="5" containerType="NO_DATA" flags="0x46
        (HAS_PRIORITY|STREAMING|HAS_QOS)" Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0
        priorityClass="1" priorityCount="1" dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1257" name="EXAMPLE.N"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 15. Market Price Request Message Sample XML Message Layout

6.4.2 Market Price Refresh Message Sample XML

```
<REFRESH domainType="MARKET_PRICE" streamId="5" containerType="FIELD_LIST" flags="0x1EA
        (HAS_PERM_DATA|HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|HAS_QOS|CLEAR_CACHE)" groupId="2"
        permData="030A CB65 62C0" Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0
        State: Open/Ok/None - text: "All is well" dataSize="1466">
    <key flags="0x03 (HAS_SERVICE_ID|HAS_NAME)" serviceId="1257" name="EXAMPLE.N"/>
    <dataBody>
        <fieldList flags="0x09 (HAS_FIELD_LIST_INFO|HAS_STANDARD_DATA)" fieldListNum="79"
                dictionaryId="1">
            <fieldEntry fieldId="3" dataType="RMTES_STRING" data="Refinitiv"/>
            <fieldEntry fieldId="22" dataType="REAL" data="30.09"/>
            <fieldEntry fieldId="25" dataType="REAL" data="30.1"/>
            <fieldEntry fieldId="26" dataType="REAL" data="30.1"/>
            <fieldEntry fieldId="30" dataType="REAL" data="4.0"/>
            <fieldEntry fieldId="31" dataType="REAL" data="2.0"/>
            <!-- Additional entries... -->
        </fieldList>
    </dataBody>
</REFRESH>
```

Figure 16. Market Price Refresh Message Sample XML Message Layout

7 Market By Order Domain

7.1 Description

The ***Market By Order*** domain provides access to Level II full order books. The list of orders is sent in the form of a **`Map`**. Each **`MapEntry`** represents one order (using the order's Id as its key) and contains a **`FieldList`** describing information related to that order (such as price, whether it is a bid/ask order, size, quote time, and market maker identifier).

NOTE: **`GenericMsg`**(s) are not supported for **DomainTypes.MARKET_BY_ORDER** Refinitiv Domain Models.

7.2 Usage

7.2.1 Market By Order Request Message

A Market By Order request message is encoded and sent by Open Message Model consumer applications. The request specifies the name of the item in which a consumer is interested.

To receive updates, the consumer makes a "streaming" request by setting the **RequestMsgFlags.STREAMING** flag. If the flag is not set, the consumer is requesting a "snapshot," and the final part of the refresh (i.e., the refresh has the **RefreshMsgFlags.REFRESH_COMPLETE** flag set) indicates all responses have been received for the snapshot. Updates may be received in either case if the refresh has multiple parts.

To stop updates, a consumer can pause an item if the provider supports this functionality. For additional details, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required. MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.MARKET_BY_ORDER == 7**
qos	Optional. Indicates the QoS at which the consumer wants the stream serviced. If both **`qos`** and **`worstQos`** are specified, this request can be satisfied by a range of qualities of service.
worstQos	Optional. Used with the **`qos`** member to define a range of acceptable Qualities of Service. When encountering such a range, the provider should attempt to provide the best QoS it can within that range. This should only be used on services that claim to support it via the **`SupportsQosRange`** item in the Source Directory response (refer to Section 4.3.1.1).
priorityClass	Optional. Indicates the class of a streams priority.
priorityCount	Optional. Indicates the count associated with a streams priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. This should be the ID associated with the service from which the consumer wants to request the item.
msgKey.nameType	Optional. When consuming from Refinitiv sources, **`msgKey.nameType`** is typically set to **InstrumentNameTypes.RIC** (the "Reuters Instrument Code"). If absent, the Enterprise Transport API assumes a setting of **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request**, otherwise optional. Specifies the requested item's name.
msgKey.filter	Not used.

Table 56: Market By Order Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Optional. When features such as View (**RequestMsgFlags.HAS_VIEW**) or Batch (**RequestMsgFlags.HAS_BATCH**) are leveraged, the payload can contain information relevant to that feature. For more detailed information, refer to *Enterprise Transport API Java Edition Developers Guide*.

Table 56: Market By Order Request Message Member Use (Continued)

7.2.2 Market By Order Refresh Message

A Market By Order refresh message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. A Market By Order refresh may be sent in multiple parts. It is possible for update and status messages to be delivered between parts of a refresh message, regardless of whether the request is streaming or non-streaming.

Providers must set the **`RefreshMsgFlags.CLEAR_CACHE`** flag on the solicited **`RefreshMsg`**. For multi-part refreshes, the **`RefreshMsgFlags.CLEAR_CACHE`** flag must be set on the first part only.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required. MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.MARKET_BY_ORDER == 7**
state	**Required**. The state of the stream and data.
partNum	Optional. Specifies the part number of a multi-part refresh.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies permission information associated with content on this stream.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`nameType`** should match the **`nameType`** specified in the request. If absent, **`msgKey.nameType`** is assumed to be **InstrumentNameTypes.RIC**.
msgKey.name	Optional. **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified and the contents of **`name`** should match the requested item's name.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. An order book is represented by an **`Map`**, where each entry (**`MapEntry`**) contains information (**`FieldList`**) that corresponds to an order.

Table 57: Market By Order Refresh Message Member Use

7.2.3 Market By Order Update Message

A Market By Order update message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. The provider can send an update message to add, update, or remove order information.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.MARKET_BY_ORDER == 7**
updateType	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **UpdateEventTypes.UNSPECIFIED == 0** • **UpdateEventTypes.QUOTE == 1**
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
conflationCount	Optional. If a provider sends a conflated update, **`conflationCount`** informs the consumer as to how many updates were included in the conflation. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
conflationTime	Optional. If a provider sends a conflated update, **`conflationTime`** informs the consumer as to the interval (in milliseconds) over which data was conflated. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
permData	Optional. **`permData`** contains permissioning information associated only with the contents of this update.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.serviceId`** specifies the ID of the service that provides the data.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** must match the name type in the item's request message (typically **InstrumentNameTypes.RIC**).
msgKey.name	Optional (Required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request). **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The order book is represented by an **`Map`**, where each map entry (**`MapEntry`**) holds information (**`FieldList`**) corresponding to an order.

Table 58: Market By Order Update Message Member Use

7.2.4 Market By Order Status Message

A Market By Order status message is sent by Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.MARKET_BY_ORDER == 7**
state	Optional. Specifies the current state information associated with the data and stream.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	Optional. The provider may use this to change the item's **`groupId`** (for details, refer to Section 4.3.1.3).
permData	Optional. **`permData`** specifies any new permissioning information associated with all of the stream's contents.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request). **`msgKey.serviceId`** specifies the ID of the service that provides the item.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** must match the name type in the item's request message. If not specified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional (Required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request). **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 59: Market By Order Status Message Member Use

7.2.5 Market By Order Post Message

If support is specified by the provider, consumer applications can post Market By Order data. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

7.3 Data

7.3.1 Market By Order Refresh / Update Payload

The payload of a Market By Order Refresh or Update is a **`Map`**. Each **`MapEntry`** corresponds to one order, where the entry key is the Order Id.

KEY TYPE	CONTAINER TYPE	PERMISSION DATA	REQUIRED	DESCRIPTION
DataTypes.BUFFER, DataTypes.ASCII_STRING, DataTypes.RMTES_STRING	DataTypes.FIELD_LIST	Optional	Yes	Contains information on the requested order book. Each **`MapEntry`** contains information about a specific order and uses the Order's ID as its key. Instead of each entry having a copy of the value in its field list, the **`keyFieldId`** of the **`Map`** can match the **`fieldId`** corresponding to the ORDER_ID field.

Table 60: Market By Order Map

7.3.2 Summary Data

The **summaryData** of the **Map** only needs to be present for the first refresh part. Typical fields in the **summaryData** include:

- Permission information (**PROD_PERM**)
- Currency of the orders (**CURRENCY**)
- Trade Units for the precision at which order prices are set (**TRD_UNITS**)
- Market State (**MKT_ST_IND**)
- Identifier of the exchange on which the orders were placed (**RDN_EXCHD2**)
- Price Ranking Rules (**PR_RNK_RUL**)
- Order Ranking Rules (**OR_RNK_RUL**)
- Quote Date (**ACTIV_DATE**)
- RIC of the underlying equity (**STOCK_RIC**)

7.3.3 MapEntry Contents

Each **MapEntry** contains a **FieldList**. Each field list contains information about the order. The field list should be decoded using its associated Field Dictionary, indicated by the **dictionaryId** present in the field list.

- For more information, refer to Section 5.2.
- For more information about use of the **Map** and **FieldList** container types, refer to the *Enterprise Transport API Java Edition Developers Guide*.

The content of each field list typically includes:

- Order Price and Side (**BID**, **ASK**, or **ORDER_PRC** and **ORDER_SIDE**)
- Order Size (**BIDSIZE**, **ASKSIZE**, or **ORDER_SIZE**)
- Price Qualifiers (**PRC_QL_CD**, **PRC_QL2**)
- Market Maker Identifier (**MKT_MKR_ID** or **MMID**)
- Quote Time (**QUOTIM_MS**)

7.4 Market By Order Sample XML

7.4.1 Market By Order Request Message Sample XML

```
<REQUEST domainType="MARKET_BY_ORDER" streamId="6" containerType="NO_DATA" flags="0x46
        (HAS_PRIORITY|STREAMING|HAS_QOS)" Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0
        priorityClass="1" priorityCount="1" dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="EXAMPLE.ARC"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 17. Market By Order Request Message Sample XML Message Layout

7.4.2 Market By Order Refresh Message Sample XML

```
<REFRESH domainType="MARKET_BY_ORDER" streamId="6" containerType="MAP" flags="0x1E8
        (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|HAS_QOS|CLEAR_CACHE)" groupId="0"
        Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0 State: Open/Ok/None -
        text: "Item Refresh Completed" dataSize="158">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="EXAMPLE.ARC"
            nameType="1"/>
    <dataBody>
        <map flags="0x13 (HAS_SET_DEFS|HAS_SUMMARY_DATA|HAS_KEY_FIELD_ID)" countHint="0"
                keyPrimitiveType="BUFFER" containerType="FIELD_LIST" keyFieldId="0" >
            <fieldSetDefs>
                <fieldSetDef setId="0">
                    <fieldSetDefEntry fieldId="3427" dataType="REAL" />
                    <fieldSetDefEntry fieldId="3428" dataType="ENUM" />
                    <fieldSetDefEntry fieldId="3429" dataType="REAL" />
                    <fieldSetDefEntry fieldId="212" dataType="RMTES_STRING" />
                    <fieldSetDefEntry fieldId="3855" dataType="UINT" />
                </fieldSetDef>
            </fieldSetDefs>
            <summaryData>
                <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                    <fieldEntry fieldId="15" dataType="ENUM" data="840"/>
                    <fieldEntry fieldId="133" dataType="ENUM" data="20"/>
                </fieldList>
            </summaryData>
            <mapEntry flags="0x00" action="ADD" key="100" >
                <fieldList flags="0x07 (HAS_FIELD_LIST_INFO|HAS_SET_DATA|HAS_SET_ID)" dictionaryId="1"
                        setId="0">
                    <fieldEntry fieldId="3427" dataType="REAL" data="34.59"/>
                    <fieldEntry fieldId="3428" dataType="ENUM" data="1"/>
                    <fieldEntry fieldId="3429" dataType="REAL" data="5.0"/>
                    <fieldEntry fieldId="212" dataType="RMTES_STRING" data="MarketMaker1"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="0"/>
                </fieldList>
            </mapEntry>
            <mapEntry flags="0x00" action="ADD" key="101" >
                <fieldList flags="0x07 (HAS_FIELD_LIST_INFO|HAS_SET_DATA|HAS_SET_ID)" dictionaryId="1"
                    setId="0">
                    <fieldEntry fieldId="3427" dataType="REAL" data="35.59"/>
                    <fieldEntry fieldId="3428" dataType="ENUM" data="2"/>
                    <fieldEntry fieldId="3429" dataType="REAL" data="6.0"/>
                    <fieldEntry fieldId="212" dataType="RMTES_STRING" data="MarketMaker2"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="500"/>
                </fieldList>
            </mapEntry>
            <mapEntry flags="0x00" action="ADD" key="102" >
                <fieldList flags="0x07 (HAS_FIELD_LIST_INFO|HAS_SET_DATA|HAS_SET_ID)" dictionaryId="1"
                        setId="0">
                    <fieldEntry fieldId="3427" dataType="REAL" data="36.59"/>
```

```
                    <fieldEntry fieldId="3428" dataType="ENUM" data="2"/>
                    <fieldEntry fieldId="3429" dataType="REAL" data="7.0"/>
                    <fieldEntry fieldId="212" dataType="RMTES_STRING" data="MarketMaker3"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="1000"/>
                </fieldList>
            </mapEntry>
        </map>
    </dataBody>
</REFRESH>
```

Figure 18. Market By Order Refresh Message Sample XML Message Layout

8 Market By Price Domain

8.1 Description

Market By Price provides access to Level II market depth information. The list of price points is sent in a **`Map`**. Each entry represents one price point (using that price and bid/ask side as its key) and contains a **`FieldList`** that describes information related to that price point.

NOTE: **`GenericMsg`**(s) are not supported for the **DomainTypes.MARKET_BY_PRICE** Refinitiv Domain Model.

8.2 Usage

8.2.1 Market By Price Request Message

A Market By Price request message is encoded and sent by Open Message Model consumer applications. The request specifies the name of an item in which the consumer is interested.

To receive updates, a consumer can make a “streaming” request by setting the **RequestMsgFlags.STREAMING** flag. If the flag is not set, the consumer requests a “snapshot” and the refresh should end the request (updates may be received in either case if the refresh has multiple parts).

A consumer can pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.MARKET_BY_PRICE == 8**
qos	Optional. Indicates the QoS at which the consumer wants the stream serviced. If both **`qos`** and **`worstQos`** are specified, this request can be satisfied by a range of QoS.
worstQos	Optional. Used with **`qos`** to define a range of acceptable QoS. When the provider encounters such a range, it should attempt to provide the best QoS possible within that range. This should only be used on services that claim to support it via the **`SupportsQosRange`** item in the Source Directory response (for further details, refer to Section 4.3.1.1).
priorityClass	Optional. Indicates the class of a streams priority.
priorityCount	Optional. Indicates the count associated with a streams priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the requested item.
msgKey.nameType	Optional. Typically set to **InstrumentNameTypes.RIC** (the “Reuters Instrument Code”) when consuming from Refinitiv sources. If absent, its default value is **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request**, otherwise optional. Specifies the name of the requested item.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.

Table 61: Market By Price Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
Payload	Optional. When features such as View (**RequestMsgFlags.HAS_VIEW**) or Batch (**RequestMsgFlags.HAS_BATCH**) are leveraged, the payload can contain information relevant to that feature. For further details, refer to the *Enterprise Transport API Java Edition Developers Guide*.

Table 61: Market By Price Request Message Member Use (Continued)

8.2.2 Market By Price Refresh Message

A Market By Price refresh message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications.

A Market By Price refresh may be sent in multiple parts. Both update and status messages can be delivered between parts of a refresh message, regardless of streaming or non-streaming request.

Providers must set the **`RefreshMsgFlags.CLEAR_CACHE`** flag on the solicited **`RefreshMsg`**. For multi-part refreshes, the **`RefreshMsgFlags.CLEAR_CACHE`** flag must be set on the first part only.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.MARKET_BY_PRICE == 8**
state	**Required**. Indicates the state of the stream and data.
partNum	Optional. Specifies the part number of a multi-part refresh.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (for further information, refer to Section 4.3.1.3).
permData	Optional. If present, specifies permission information associated with the stream's content.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`msgKey.nameType`** should match the **`msgKey.nameType`** specified in the request. If absent, this value is assumed to be **InstrumentNameTypes.RIC**.
msgKey.name	Optional. **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified, and **`msgKey.name`** should match the name specified in the request.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The order book is represented by a **`Map`**, where each entry contains a **`FieldList`** which has information about a price point.

Table 62: Market By Price Refresh Message Member Use

8.2.3 Market By Price Update Message

A Market By Price update message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. The provider can send an update message to add, update, or remove price point information.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.MARKET_BY_PRICE == 8**
updateType	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **UpdateEventTypes.UNSPECIFIED == 0** • **UpdateEventTypes.QUOTE == 1**
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
conflationCount	Optional. If a provider sends a conflated update, **`conflationCount`** specifies how many updates were included in the conflation. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** to **true** in the request.
conflationTime	Optional. If a provider sends a conflated update, **`conflationTime`** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** to **true** in the request.
permData	Optional. Specifies permissioning information for the update's content.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the ID of the service that provides the item.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** should match the **`msgKey.nameType`** specified in the item's request message. If **`msgKey.nameType`** is not specified, it uses the default **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **`msgKey.name`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request) Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The order book is represented by a **`Map`**, where each entry contains a **`FieldList`** containing information about a price point.

Table 63: Market By Price Update Message Member Use

8.2.4 Market By Price Status Message

A Market By Price status message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.MARKET_BY_PRICE == 8**
state	Optional. Specifies current state information associated with the data and stream.
qos	Optional. Specifies the QoS at which the stream is provided.
groupId	Optional. Specifies the item's **`groupId`** (the provider can use this component to change the item's **`groupId`**).
permData	Optional. Specifies new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
msgKey.serviceId	Optional. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`msgKey.nameType`** should match the **`msgKey.nameType`** specified in the item's request message. If **`msgKey.nameType`** is not specified, it uses the default **InstrumentNameTypes.RIC**.
msgKey.name	Optional. Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 64: Market By Price Status Message Member Use

8.2.5 Market By Price Post Message

If supported by the provider, consumer applications can post Market By Price data. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

8.3 Data

8.3.1 Market By Price Refresh/Update Payload

The payload of a Market By Price Refresh or Update is a **Map**. Each price point is contained in an **MapEntry** which uses the price point and side (buy or sell) as the entry key.

KEY TYPE	CONTAINER TYPE	PERMISSION DATA	DESCRIPTION
DataTypes.BUFFER	DataTypes.FIELD_LIST	Optional	**Required**. Contains information about the known price points. Each **MapEntry** represents one price point and uses that price (along with the buy/sell side) as its key.

Table 65: Market By Price Map

8.3.2 Summary Data

The **summaryData** of the **Map** needs to be present only for the first refresh part, which typically includes:

- Permission information (**PROD_PERM**)
- Currency of the orders (**CURRENCY**)
- Trade Units for the precision with which order prices are set (**TRD_UNITS**)
- Market State (**MKT_ST_IND**)
- The identifier of the exchange on which the orders were placed (**RDN_EXCHD2**)
- Price Ranking Rules (**PR_RNK_RUL**)
- Quote Date (**QUOTE_DATE**)

8.3.3 MapEntry Contents

The **MapEntry** key is a **Buffer** that contains a combination of the price and order side, thus each key is unique within its map. The key should be treated as a single entity and is not meant to be parsed.

Each **MapEntry** houses an **FieldList** that contains information about the price point. The field list should be decoded using its associated Field Dictionary, indicated by the **dictionaryId** present in the field list.

- For more information on dictionary use, refer to Section 5.2.
- For more information about use of the **Map** and **FieldList** container types, refer to the *Enterprise Transport API Java Edition Developers Guide*.

The field list typically includes:

- Order Price & Side (**BID**, **ASK**, or **ORDER_PRC** and **ORDER_SIDE**)
- Order Size (**BIDSIZE**, **ASKSIZE**, or **ORDER_SIZE**)
- Number of aggregated orders (**NO_ORD**)
- Quote Time (**QUOTIM_MS**)
- A map containing the Market Makers (**MMID**) and optionally a field list with each of the market makers' positions at the Order Price point.

8.4 Market By Price Sample XML

8.4.1 Market By Price Request Message Sample XML

```
<REQUEST domainType="MARKET_BY_PRICE" streamId="7" containerType="NO_DATA" flags="0x46
        (HAS_PRIORITY|STREAMING|HAS_QOS)" Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0
        priorityClass="1" priorityCount="1" dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="EXAMPLE.N"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 19. Market By Price Request Message Sample XML Message Layout

8.4.2 Market By Price Refresh Message Sample XML

```
<REFRESH domainType="MARKET_BY_PRICE" streamId="7" containerType="MAP" flags="0x1A8
        (HAS_MSG_KEY|SOLICITED|HAS_QOS|CLEAR_CACHE)" groupId="0" Qos: Realtime/TickByTick/Static -
        timeInfo: 0 - rateInfo: 0 State: Open/Ok/None - text: "Item Refresh In Progress" dataSize="59">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="EXAMPLE.N"
            nameType="1"/>
    <dataBody>
        <map flags="0x13 (HAS_SET_DEF|HAS_SUMMARY_DATA|HAS_KEY_FIELD_ID)" countHint="0"
                keyPrimitiveType="BUFFER" containerType="FIELD_LIST" keyFieldId="0" >
        <fieldSetDefs>
            <fieldSetDef setId="0">
                <fieldSetDefEntry fieldId="3430" dataType="UINT" />
                <fieldSetDefEntry fieldId="3428" dataType="ENUM" />
                <fieldSetDefEntry fieldId="3429" dataType="REAL" />
                <fieldSetDefEntry fieldId="3427" dataType="REAL" />
                <fieldSetDefEntry fieldId="3855" dataType="UINT" />
            </fieldSetDef>
        </fieldSetDefs>
        <summaryData>
            <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                    <fieldEntry fieldId="15" dataType="ENUM" data="840"/>
                    <fieldEntry fieldId="133" dataType="ENUM" data="20"/>
                </fieldList>
            </summaryData>
            <mapEntry flags="0x00" action="ADD" key="34.59b" >
                <fieldList flags="0x07 (HAS_FIELD_LIST_INFO|HAS_SET_DATA|HAS_SET_ID)"
                        dictionaryId="1" setId="0">
                    <fieldEntry fieldId="3430" dataType="UINT" data="1"/>
                    <fieldEntry fieldId="3428" dataType="REAL" data="5.0"/>
                    <fieldEntry fieldId="3429" dataType="REAL" data="5.0"/>
```

```
                    <fieldEntry fieldId="3427" dataType="REAL" data="34.59"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="0"/>
                </fieldList>
            </mapEntry>
        </map>
    </dataBody>
</REFRESH>
```

Figure 20. Market By Price Refresh Message Sample XML Message Layout

9 Market Maker Domain

9.1 Description

The ***Market Maker*** domain provides access to market maker quotes and trade information. The list of market makers is sent in the form of a **`Map`**. Each **`MapEntry`** represents one market maker (using that market maker's ID as its key) and contains a **`FieldList`** describing information such as that market maker's bid and ask prices, quote time, and market source.

NOTE: **`GenericMsg`**(s) are not supported for the **MARKET_MAKER** Refinitiv Domain Model.

9.2 Usage

9.2.1 Market Maker Request Message

A Market Maker request message is encoded and sent by Open Message Model consumer applications. The request specifies the name of an item in which the consumer is interested.

To receive updates, a consumer can make a "streaming" request by setting the **RequestMsgFlags.STREAMING** flag. If the flag is not set, the consumer requests a "snapshot," and the final part of the refresh (i.e., the refresh has the **RefreshMsgFlags.REFRESH_COMPLETE** flag set) indicates all responses have been received for the snapshot. Updates may be received in either case if the refresh has multiple parts.

To stop updates, a consumer can pause an item (if the provider supports this functionality). For more information, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.MARKET_MAKER == 9**
qos	Optional. Indicates the QoS at which the consumer wants the stream serviced. If both **`qos`** and **`worstQos`** are specified, this request can be satisfied by a range of QoS.
worstQos	Optional. Used with **`qos`** to define a range of acceptable QoS. If the provider encounters such a range, it should attempt to provide the best possible QoS within that range. This should only be used on services that claim to support it via the **`SupportsQosRange`** item in the Source Directory response (for details, refer to Section 4.3.1.1).
priorityClass	Optional. Indicates the class of a stream's priority.
priorityCount	Optional. Indicates the count associated with a stream's priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the requested item.
msgKey.nameType	Optional. When consuming from Refinitiv sources, **`msgKey.nameType`** is typically set to **InstrumentNameTypes.RIC** (the "Reuters Instrument Code"). If absent, its value reverts to the default, which is **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request**, otherwise optional. Specifies the name of the requested item.
msgKey.filter	Not used.
msgKey.identifier	Not used.

Table 66: Market Maker Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.attrib	Not used.
Payload	Optional. When features such as View (**RequestMsgFlags.HAS_VIEW**) or Batch (**RequestMsgFlags.HAS_BATCH**) are leveraged, the payload can contain information relevant to that feature. For more details, refer to *Enterprise Transport API Java Edition Developers Guide*.

Table 66: Market Maker Request Message Member Use (Continued)

9.2.2 Market Maker Refresh Message

A Market Maker refresh message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications.

The Market Maker refresh can be sent in multiple parts. Keep in mind that both update and status messages can be delivered between parts of a refresh message, regardless of streaming or non-streaming request.

Providers must set the **`RefreshMsgFlags.CLEAR_CACHE`** flag on the solicited **`RefreshMsg`**. For multi-part refreshes, the **`RefreshMsgFlags.CLEAR_CACHE`** flag must be set on the first part only.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.MARKET_MAKER == 9**
state	**Required**. Indicates the state of the stream and data.
partNum	Optional. Specifies the part number of a multi-part refresh.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies permission information associated with this stream's content.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`nameType`** should match the **`nameType`** specified in the request. If absent, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. A **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified, which should match the requested **`name`**.
msgKey.filter	Not used.
msgKey.identifier	Not used.
Payload	**Required**. The order book is represented by a **`Map`**, where each entry contains an **`FieldList`** which has information about a market maker.

Table 67: Market Maker Refresh Message Member Use

9.2.3 Market Maker Update Message

A Market Maker update message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications.

The provider can send an update message to add, update, or remove market maker information.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.MARKET_MAKER == 9**
updateType	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **UpdateEventTypes.UNSPECIFIED == 0** • **UpdateEventTypes.QUOTE == 1**
partNum	Not used.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
conflationCount	Optional. If a provider sends a conflated update, **`conflationCount`** specifies how many updates are in the conflation. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
conflationTime	Optional. If a provider sends a conflated update, **`conflationTime`** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
permData	Optional. Specifies permissioning information associated only with the contents of this update.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.serviceId`** specifies the ID of the service that provides the item.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.nameType`** must match the name type in the item's request message (typically **InstrumentNameTypes.RIC**). If absent, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **`msgKey.name`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The order book is represented by a **`Map`**, where each entry contains a **`FieldList`** which in turn contains information about a market maker.

Table 68: Market Maker Update Message Member Use

9.2.4 Market Maker Status Message

A Market Maker status message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.MARKET_MAKER == 9**
state	Optional. Specifies current state information associated with the data and stream.
qos	Optional. Specifies the QoS at which the stream is provided.
groupId	Optional. The provider can use this component to change the items' **`groupId`**.
permData	Optional. Specifies new permissioning information associated with all of the stream's contents.
extendedHeader	Not used.
msgKey.serviceId	Optional. **`msgKey.serviceId`** specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`msgKey.nameType`** must match the name type in the item's request message (typically **InstrumentNameTypes.RIC**). If absent, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. **`msgKey.name`** specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 69: Market Maker Status Message Member Use

9.2.5 Market Maker Post Message

If the provider supports Market Maker post messages, consumer applications can post Market Maker data. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

9.3 Data

9.3.1 Market Maker Refresh/Update Payload

The payload of a Market Maker Refresh or Update is an **Map**. Information about each market maker is contained in an **MapEntry** which uses the market maker ID as its entry key.

KEY TYPE	CONTAINER TYPE	DESCRIPTION
DataTypes.BUFFER DataTypes.ASCII_STRING DataTypes.RMTES_STRING	DataTypes.FIELD_LIST	**Required**. Contains information about known market makers. Each **MapEntry** identifies one market maker and uses the market maker's ID as the key. Including permission data is optional. The **keyFieldId** of the map may be set to the **fieldId** that corresponds to the **MMID** or **MKT_MKR_ID** field, instead of each entry having a copy of the value in its field list.

Table 70: Market Maker Map

9.3.2 Summary Data

The **summaryData** of the **Map** only needs to be present in the first refresh part. Typical fields in the **summaryData** include:

- Permission information (**PROD_PERM**)
- Currency of the orders (**CURRENCY**)
- Trade Units for the precision at which order prices are set (**TRD_UNITS**)
- Identifier of the exchange on which the orders were placed (**RDN_EXCHD2**)
- Market State indicating the state of the market (**MKT_ST_IND**)
- Price ranking rules (**PR_RNK_RUL**)
- Quote Date (**QUOTE_DATE**)

9.3.3 MapEntry Contents

Each **MapEntry** key is a **Buffer**, **AsciiString**, or **RmtesString** that contains a unique market maker's ID.

Each **MapEntry** houses a **FieldList** that contains information about the market maker. The field list should be decoded using its associated Field Dictionary, indicated by the **dictionaryId** present in the field list.

- For more information on dictionary use, refer to Section 5.2.
- For more information about use of the **Map** and **FieldList** container types, refer to the *Enterprise Transport API Java Edition Developers Guide*.

The field list typically includes:

- Bid (**BID**)
- Ask (**ASK**)
- Bid Size (**BIDSIZE**)
- Ask Size (**ASKSIZE**)
- Market Source (**MKT_SOURCE**)
- Market Maker Name (**MKT_MKR_NM**)
- Price Qualifiers (**PRC_QL_CD** and **PRC_QL2**)

- Quote Time (**QUOTIM_MS**)

9.4 Market Maker Sample XML

9.4.1 Market Maker Request Message Sample XML

```
<REQUEST domainType="MARKET_MAKER" streamId="6" containerType="NO_DATA" flags="0x06
        (HAS_PRIORITY|STREAMING)" priorityClass="1" priorityCount="1" dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="EXAMPLE.OQ"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 21. Market Maker Request Message Sample XML Message Layout

9.4.2 Market Maker Refresh Message Sample XML

```
<REFRESH domainType="MARKET_MAKER" streamId="6" containerType="MAP" flags="0x128
        (HAS_MSG_KEY|SOLICITED|CLEAR_CACHE)" groupId="0" State: Open/Ok/None - text: "OK"
        dataSize="560">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="TRI.N" nameType="1"/
            >
    <dataBody>
        <map flags="0x03 (HAS_SET_DEFS|HAS_SUMMARY_DATA)" countHint="0" keyPrimitiveType="BUFFER"
                containerType="FIELD_LIST" >
            <fieldSetDefs>
                <fieldSetDef setId="0">
                    <fieldSetDefEntry fieldId="22" dataType="REAL" />
                    <fieldSetDefEntry fieldId="30" dataType="REAL" />
                    <fieldSetDefEntry fieldId="25" dataType="REAL" />
                    <fieldSetDefEntry fieldId="31" dataType="REAL" />
                    <fieldSetDefEntry fieldId="213" dataType="ENUM" />
                    <fieldSetDefEntry fieldId="214" dataType="RMTES_STRING" />
                    <fieldSetDefEntry fieldId="3855" dataType="UINT" />
                    <fieldSetDefEntry fieldId="118" dataType="ENUM" />
                </fieldSetDef>
            </fieldSetDefs>
            <summaryData>
                <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                    <fieldEntry fieldId="15" dataType="ENUM" data="840"/>
                    <fieldEntry fieldId="1709" dataType="ENUM" data="2"/>
                    <fieldEntry fieldId="53" dataType="ENUM" data="2"/>
                    <fieldEntry fieldId="3423" dataType="ENUM" data="2"/>
                    <fieldEntry fieldId="133" dataType="ENUM" data="1"/>
                </fieldList>
```

```
            </summaryData>
            <mapEntry flags="0x00" action="ADD" key="MMID1" >
                <fieldList flags="0x06 (HAS_SET_DATA|HAS_SET_ID)" setId="0">
                    <fieldEntry fieldId="22" dataType="REAL" data="14.72"/>
                    <fieldEntry fieldId="30" dataType="REAL" data="165.0"/>
                    <fieldEntry fieldId="25" dataType="REAL" data="17.28"/>
                    <fieldEntry fieldId="31" dataType="REAL" data="110.0"/>
                    <fieldEntry fieldId="213" dataType="ENUM" data="1"/>
                    <fieldEntry fieldId="214" dataType="RMTES_STRING" data="Market Maker1(0x00)"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="68627452"/>
                    <fieldEntry fieldId="118" dataType="ENUM" data="60"/>
                </fieldList>
            </mapEntry>
            <mapEntry flags="0x00" action="ADD" key="MMID2" >
                <fieldList flags="0x06 (HAS_SET_DATA|HAS_SET_ID)" setId="0">
                    <fieldEntry fieldId="22" dataType="REAL" data="16.0"/>
                    <fieldEntry fieldId="30" dataType="REAL" data="166.0"/>
                    <fieldEntry fieldId="25" dataType="REAL" data="16.0"/>
                    <fieldEntry fieldId="31" dataType="REAL" data="167.0"/>
                    <fieldEntry fieldId="213" dataType="ENUM" data="4"/>
                    <fieldEntry fieldId="214" dataType="RMTES_STRING" data="Market Maker2(0x00)"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="68627452"/>
                    <fieldEntry fieldId="118" dataType="ENUM" data="60"/>
                </fieldList>
            </mapEntry>
            <mapEntry flags="0x00" action="ADD" key="MMID3" >
                <fieldList flags="0x06 (HAS_SET_DATA|HAS_SET_ID)" setId="0">
                    <fieldEntry fieldId="22" dataType="REAL" data="16.0"/>
                    <fieldEntry fieldId="30" dataType="REAL" data="158.0"/>
                    <fieldEntry fieldId="25" dataType="REAL" data="16.0"/>
                    <fieldEntry fieldId="31" dataType="REAL" data="137.0"/>
                    <fieldEntry fieldId="213" dataType="ENUM" data="3"/>
                    <fieldEntry fieldId="214" dataType="RMTES_STRING" data="Market Maker3(0x00)"/>
                    <fieldEntry fieldId="3855" dataType="UINT" data="68627452"/>
                    <fieldEntry fieldId="118" dataType="ENUM" data="60"/>
                </fieldList>
            </mapEntry>
        </map>
    </dataBody>
</REFRESH>
```

Figure 22. Market Maker Refresh Message Sample XML Message Layout

10 Yield Curve Domain

10.1 Description

The ***Yield Curve*** domain shows the relation between the interest rate and the term (time to maturity) associated with the debt of a borrower. The shape of a yield curve can help give an idea of future economic activity and interest rates. Information is sent as a **`FieldList`**, where some **`FieldEntry`**'s can contain more complex types such as **`Vector`**, **`Array`**, or **`ElementList`**.

This chapter documents the Yield Curve domain as provided by the Refinitiv Real-Time Advanced Transformation Server.

NOTE: The **YIELD_CURVE** Refinitiv Domain Model does not support **`GenericMsg`**(s).

10.2 Usage

10.2.1 Yield Curve Request Message

A Yield Curve request message is encoded and sent by Open Message Model consumer applications. The request specifies the name and attributes of the curve in which the consumer is interested.

To receive updates, the consumer makes a "streaming" request by setting the **RequestMsgFlags.STREAMING** flag. If the flag is not set, the consumer requests a "snapshot," and the final part of the refresh (i.e., the refresh has the **RefreshMsgFlags.REFRESH_COMPLETE** flag set) indicates all responses have been received for the snapshot. Updates may be received in either case if the refresh has multiple parts.

To stop updates, a consumer can pause an item (if the provider supports the pause feature). For additional details, refer to the *Enterprise Transport API Java Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
msgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.YIELD_CURVE == 22**
qos	Optional. Indicates the QoS at which the consumer wants the stream serviced. If both **`qos`** and **`worstQos`** are specified, this request can be satisfied by a range of QoS.
worstQos	Optional. Used with the **`qos`** member to define a range of acceptable QoS. When the provider encounters such a range, it should attempt to provide the best QoS it can within that range. **`worstQos`** should only be used on services that claim to support it via the **`SupportsQosRange`** item in the Source Directory response (refer to Section 4.3.1.1).
priorityClass	Optional. Indicates the class of a streams priority.
priorityCount	Optional. Indicates the count associated with a streams priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the requested item.
msgKey.nameType	Optional. When consuming from Refinitiv sources, typically set to **InstrumentNameTypes.RIC** (the "Reuters Instrument Code"). If this is not specified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request**, otherwise optional.Specifies the name of the requested item.
msgKey.filter	Not used.
msgKey.identifier	Not used.

Table 71: Yield Curve Request Message Member Use

COMPONENT	DESCRIPTION / VALUE
msgKey.attrib	Not used.
Payload	Optional. When leveraging such features as View (**RequestMsgFlags.HAS_VIEW**) or Batch (**RequestMsgFlags.HAS_BATCH**), the payload can contain information relevant to that feature. For more information, refer to *Enterprise Transport API Java Edition Developers Guide*.

Table 71: Yield Curve Request Message Member Use (Continued)

10.2.2 Yield Curve Refresh Message

A Yield Curve Refresh Message is sent by Open Message Model provider and non-interactive provider applications. This message sends all currently available information about the item to the consumer.

FieldList in the payload should include all fields that might be present in subsequent updates, even if those fields are currently blank. When responding to a View request, this refresh should contain all fields requested by the specified view. If for any reason the provider wishes to send new fields, it must first send an unsolicited refresh with both the new and currently-present fields.

Providers must set the **RefreshMsgFlags.CLEAR_CACHE** flag on the solicited **RefreshMsg**. For multi-part refreshes, the **RefreshMsgFlags.CLEAR_CACHE** flag must be set on the first part only.

WARNING! Although the payload is an **FieldList**, some field entries are sent as more complex types such as **Vector** and **Array**. Encoding and decoding applications should be aware of this and ensure proper handling of these types.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.YIELD_CURVE == 22**
state	**Required**. Includes the state of the stream and data.
partNum	Optional. Specifies the part number of a multi-part refresh.
qos	Optional. Specifies the QoS at which the stream is provided.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies permission information associated with content on this stream.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. Should match the **nameType** specified in the request. If this is not specified, **nameType** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. **msgKey.flags** value of **MsgKeyFlags.HAS_NAME** should be specified. This should match the requested name.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. This should consist of a **FieldList** containing all fields associated with the item.

Table 72: Yield Curve Refresh Message Member Use

10.2.3 Yield Curve Update Message

A Yield Curve Update Message is sent by Open Message Model provider and non-interactive provider applications. It conveys any changes to an item's data.

WARNING! Although the payload is an **FieldList**, some field entries are sent as more complex types such as **Vector** and **Array**. Encoding and decoding applications should be aware of this and ensure proper handling of these types.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.YIELD_CURVE == 22**
updateType	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **UpdateEventTypes.UNSPECIFIED == 0** • **UpdateEventTypes.QUOTE == 1**
seqNum	Optional. A user-specified, item-level sequence number which the application can use to sequence messages in this stream.
partNum	Not used.
conflationCount	Optional. If the provider sends a conflated update, **conflationCount** specifies how many updates are in the conflation. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
conflationTime	Optional. If a provider is sending a conflated update, **conflationTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **RequestMsgFlags.CONF_INFO_IN_UPDATES** flag in the request.
permData	Optional. Permissioning information associated with only the contents of this update.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **msgKey.serviceId** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the ID of the service that provides the item.
msgKey.nameType	**Conditional**. **msgKey.nameType** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Should match the **msgKey.nameType** specified on the request. If this is not specified, **msgKey.nameType** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **msgKey.name** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. This should consist of a **FieldList** with any changed data.

Table 73: Yield Curve Update Message Member Use

10.2.4 Yield Curve Status Message

A Yield Curve status message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.YIELD_CURVE == 22**
state	Optional. Current state information associated with the data and stream.
qos	Optional. Specifies the QoS at which the stream is provided.
groupId	Optional. The provider can use this component to change the item's **`groupId`**.
permData	Optional. Specifies new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
msgKey.serviceId	Optional. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. Should match the **`msgKey.nameType`** specified on the request. If this is not specified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 74: Yield Curve Status Message Member Use

10.2.5 Yield Curve Domain Post Message

If supported by the provider, consumer applications can post Yield Curve data. For more information on posting, refer to the *Enterprise Transport API Java Edition Developers Guide*.

10.3 Data

10.3.1 Yield Curve Refresh/Update Payload

The payload of a Yield Curve Refresh or Update is a **`FieldList`**. Some **`FieldEntry`** contents contain primitive type information to help describe the curve. Examples include the Curve Type (**CRV_TYPE**), the Algorithm used to calculate the curve (**CRV_ALGTHM**), and the Interpolation (**INTER_MTHD**) and Extrapolation (**EXTRP_MTHD**) methods.

Other **`FieldEntry`**'s contain more complex information. The more complex entries are broken down into:

- Input Entries which define the different input data used to calculate the yield curve. Inputs are represented using non-sorted **`Vector`** types. Examples of curve inputs would be cash rates (**CASH_RATES**), future prices (**FUTR_PRCS**), and swap rates (**SWAP_RATES**).
- Output Entries which define the output of the yield curve calculation. Outputs are represented using non-sorted **`Vector`** types. An example of curve outputs would be the Yield Curve (**YLD_CURVE**) itself.
- Extra Meta Data (**EX_MET_DAT**) which provides general data about the yield curve. This is represented using a **`ElementList`** type. Extra meta data allows users to provide additional curve descriptions without needing to define new fields. Some examples of meta data would be curve creation time or the curve's owner.

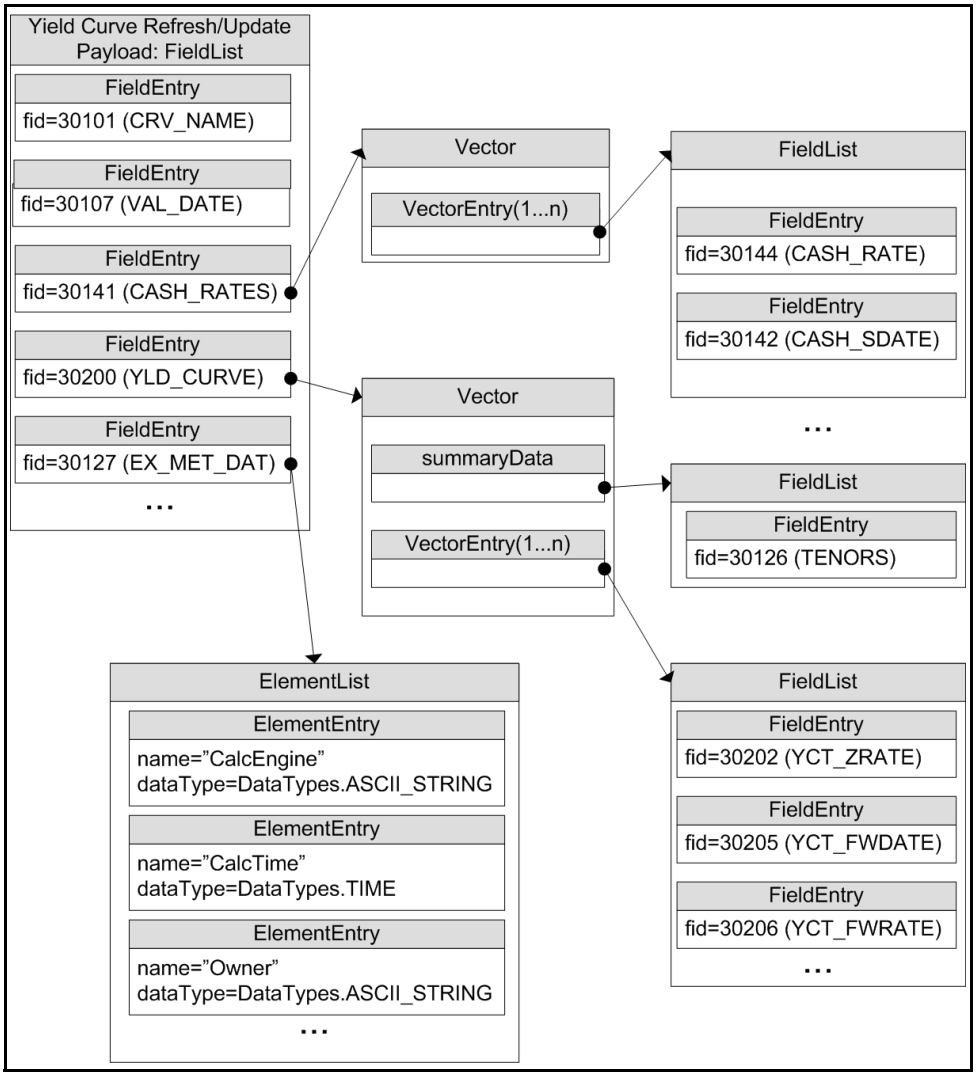


Figure 23. Yield Curve Payload Example

10.3.2 Summary Data

For **Vector** types, **summaryData** can be included to provide information specific to the **Vector**'s contents. Any **summaryData** needs to be present only for the first refresh part that contains the **Vector**. Typical **summaryData** fields include tenors (**TENORS**).

10.3.3 Yield Curve Input and Output Entries

Each **VectorEntry** houses a **FieldList** that contains specific information about the respective input or output. The field list should be decoded using its associated Field Dictionary, indicated by the **dictionaryId** present in the field list.

- For more information on dictionary use, refer to Section 5.2.
- For more information about use of the **Vector** and **FieldList** container types, refer to the *Enterprise Transport API Java Edition Developers Guide*.

The following table contains additional information about input and output entries (all of which are of the DataTypes.VECTOR container type with a container entry type of DataTypes.FIELD_LIST).

NAME	FIELD NAME	TYPE	DESCRIPTION
Cash Rates	CASH_RATES	Input	Contains cash rate data used to calculate the yield curve output. This typically includes information like settlement date (**CASH_SDATE**), maturity date (**CASH_MDATE**), and other related fields.
Future Prices	FUTR_PRCS	Input	Contains future pricing data used to calculate the yield curve output; typically including data such as settlement date (**FUTR_SDATE**), maturity date (**FUTR_MDATE**), and other related fields.
Swap Rates	SWAP_RATES	Input	Contains swap rate data used to calculate the yield curve output; typically including data such as settlement date (**SWAP_SDATE**), maturity date (**SWAP_MDATE**), swap rate value (**SWAP_RATE_VAL**), and roll date (**SWAP_RDATE**).
Spread Rates	SPRD_RATES	Input	Contains spread rate data used to calculate yield curve output; typically including data such as spread frequency (**SPRD_FREQ**), maturity date (**SPRD_MDATE**), spread rate (**SPRD_RATE**), and roll date (**SPRD_RDATE**).
Yield Curve	YLD_CURVE	Output	Contains calculated Yield Curve data; typically including data such as zero rate (**YCT_ZRATE**), forward rate (**YCT_FWRATE**), and discount factor (**YCT_DISFAC**).

Table 75: Yield Curve Inputs and Outputs

10.4 Specific Usage: ATS

When an application consumes Yield Curve data, the dictionary used by the application must contain certain required Field IDentifiers. For further details, refer to the Refinitiv Real-Time Advanced Transformation Server documentation.

10.5 Yield Curve Sample XML

10.5.1 Yield Curve Request Message Sample XML

```
<REQUEST domainType="YIELD_CURVE" streamId="6" containerType="NO_DATA" flags="0x06
        (HAS_PRIORITY|STREAMING)" priorityClass="1" priorityCount="1" dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="7264" name="YCUSD01"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 24. Yield Curve Request Message Sample XML Message Layout

10.5.2 Yield Curve Refresh Message Sample XML

```
<REFRESH domainType="YIELD_CURVE" streamId="6" containerType="FIELD_LIST" flags="0x1E8
        (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|HAS_QOS|CLEAR_CACHE)" groupId="1"
        Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0 State: Open/Ok/None -
        text: "" dataSize="647">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="7264" name="YCMAT01"
            nameType="1"/>
    <dataBody>
        <fieldList flags="0x09 (HAS_FIELD_LIST_INFO|HAS_STANDARD_DATA)" fieldListNum="0"
                dictionaryId="1">
            <fieldEntry fieldId="30100" dataType="INT" data="1"/>
            <fieldEntry fieldId="30101" dataType="ASCII_STRING" data="YCMAT01"/>
            <fieldEntry fieldId="30102" dataType="ASCII_STRING"/>
            <fieldEntry fieldId="30103" dataType="ASCII_STRING" data="Swap"/>
            <fieldEntry fieldId="30104" dataType="ASCII_STRING" data="Standard"/>
            <fieldEntry fieldId="30105" dataType="DATETIME" data="24 JAN 2013 15:11:03:000"/>
            <fieldEntry fieldId="30106" dataType="ASCII_STRING" data="US"/>
            <fieldEntry fieldId="30107" dataType="DATE" data="24 JAN 2013 "/>
            <fieldEntry fieldId="30108" dataType="DATE" data="24 JAN 2013 "/
            <fieldEntry fieldId="30109" dataType="ASCII_STRING" data="Refinitiv Advanced
                    Transformation System"/>
            <fieldEntry fieldId="30110" dataType="ASCII_STRING"/>
            <fieldEntry fieldId="30111" dataType="ASCII_STRING"/>
            <fieldEntry fieldId="30112" dataType="ASCII_STRING" data="Bootstrap"/>
            <fieldEntry fieldId="30113" dataType="ASCII_STRING" data="Modified Following"/>
            <fieldEntry fieldId="30114" dataType="ASCII_STRING" data="ACT/360"/>
            <fieldEntry fieldId="30115" dataType="INT" data="0"/>
            <fieldEntry fieldId="30116" dataType="ASCII_STRING" data="Compound"/>
            <fieldEntry fieldId="30117" dataType="ASCII_STRING" data="ACT/360"/>
            <fieldEntry fieldId="30118" dataType="ASCII_STRING" data="Created"/>
            <fieldEntry fieldId="30119" dataType="ASCII_STRING" data="ADMIN"/>
            <fieldEntry fieldId="30120" dataType="ASCII_STRING" data="ADMIN"/>
```

```
            <fieldEntry fieldId="30121" dataType="DATETIME" data="17 SEP 2012 00:00:00:000"/>
            <fieldEntry fieldId="30122" dataType="DATETIME" data="17 SEP 2012 00:00:00:000"/>
            <fieldEntry fieldId="30123" dataType="ASCII_STRING"/>
            <fieldEntry fieldId="30124" dataType="ASCII_STRING" data="ACT/360"/>
            <fieldEntry fieldId="30157" dataType="ASCII_STRING" data="Linear"/>
            <fieldEntry fieldId="16" dataType="DATE" data="24 JAN 2013 "/>
            <fieldEntry fieldId="5" dataType="TIME" data="22:11:03:000"/>
            <fieldEntry fieldId="30168" dataType="ASCII_STRING" data="ACT/360"/>
            <fieldEntry fieldId="30169" dataType="ASCII_STRING" data="Annually"/>
            <fieldEntry fieldId="30159" dataType="ASCII_STRING" data="Linear"/>
            <fieldEntry fieldId="30161" dataType="VECTOR">
                <vector flags="0x02 (HAS_SUMMARY_DATA)" countHint="0" containerType="FIELD_LIST">
                    <summaryData>
                        <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                            <fieldEntry fieldId="30126" dataType="ARRAY">
                                <array itemLength="0" primitiveType="ASCII_STRING">
                                    <arrayEntry data="3Y"/>
                                    <arrayEntry data="5Y"/>
                                    <arrayEntry data="10Y"/>
                                </array>
                            </fieldEntry>
                        </fieldList>
                    </summaryData>
                    <vectorEntry index="0" action="SET" flags="0x00">
                        <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                            <fieldEntry fieldId="30162" dataType="DATE" data="24 JAN 2013 "/>
                            <fieldEntry fieldId="30163" dataType="DATE" data="25 JAN 2016 "/>
                            <fieldEntry fieldId="30164" dataType="DATE" data="25 JAN 2016 "/>
                            <fieldEntry fieldId="30166" dataType="REAL" data="88.130997"/>
                            <fieldEntry fieldId="30167" dataType="ASCII_STRING" data="JPY3Y="/>
                        </fieldList>
                    </vectorEntry>
                    <vectorEntry index="1" action="SET" flags="0x00">
                        <fieldList flags="0x08 (HAS_STANDARD_DATA)">
                            <fieldEntry fieldId="30162" dataType="DATE" data="24 JAN 2013 "/>
                            <fieldEntry fieldId="30163" dataType="DATE" data="24 JAN 2023 "/>
                            <fieldEntry fieldId="30164" dataType="DATE" data="24 JAN 2023 "/>
                            <fieldEntry fieldId="30166" dataType="REAL" data="74.029999"/>
                            <fieldEntry fieldId="30167" dataType="ASCII_STRING" data="JPY10Y="/>
                        </fieldList>
                    </vectorEntry>
                </vector>
            </fieldEntry>
            <!-- Additional entries... -->
        </fieldList>
    </dataBody>
</REFRESH>
```

Figure 25. Yield Curve Refresh Message Sample XML Message Layout

11 Symbol List Domain

11.1 Description

The ***Symbol List*** domain provides access to a set of symbol names, typically from an index, service, or cache. Content is encoded as a **`Map`**, with each symbol represented by a map entry and where the symbol name is the entry key. An entry's payload is optional, but when present the payload is a **`FieldList`** or **`ElementList`** that contains additional cross-reference information such as permission information, name type, or other venue-specific content.

NOTE: **`GenericMsg`**(s) are not supported for **SYMBOL_LIST** Refinitiv Domain Model.

11.2 Usage

11.2.1 Symbol List Request Message

A Symbol List request message is encoded and sent by Open Message Model consumer applications.

The consumer can make a streaming request (set **RequestMsgFlags.STREAMING**) to receive updates, typically associated with item additions or removals from the list.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REQUEST == 1**
domainType	**Required**. **DomainTypes.SYMBOL_LIST == 10**
qos	Not used.
worstQos	Not used.
priorityClass	Optional. Specifies the class of a stream's priority.
priorityCount	Optional. Specifies the count associated with a stream's priority.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the requested item.
msgKey.nameType	Optional. **`msgKey.nameType`** should match name type specified in the request. When consuming from Refinitiv sources, **`msgKey.nameType`** is typically set to **InstrumentNameTypes.RIC** (the "Reuters Instrument Code"). If absent, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	**Required in initial request**, otherwise optional. Specifies the name of the requested item.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Optional. When leveraging such features as View (**RequestMsgFlags.HAS_VIEW**), Batch (**RequestMsgFlags.HAS_BATCH**), or behaviors related to the Symbol List Request, the payload can contain information relevant to that feature. For more detailed information, refer to *Enterprise Transport API Java Edition Developers Guide*.

Table 76: Symbol List Request Message Member Use

11.2.2 Symbol List Refresh Message

A Symbol List refresh Message is sent by Open Message Model provider and non-interactive provider applications. This message sends a list of item names to the consumer.

A Symbol List refresh can be sent in multiple parts. Update and status messages can be delivered between parts of a refresh message, regardless of streaming or non-streaming request.

Providers must set the **`RefreshMsgFlags.CLEAR_CACHE`** flag on the solicited **`RefreshMsg`**. For multi-part refreshes, the **`RefreshMsgFlags.CLEAR_CACHE`** flag must be set on the first part.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.REFRESH == 2**
domainType	**Required**. **DomainTypes.SYMBOL_LIST == 10**
state	**Required**. Indicates the state of the stream and data.
partNum	Optional. Specifies the part number of a multi-part refresh.
qos	Not used.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
groupId	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
permData	Optional. Specifies the permission information associated with content on this stream.
extendedHeader	Not used.
msgKey.serviceId	**Required**. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`nameType`** should match the **`nameType`** specified in the request. If absent, it is assumed to be **InstrumentNameTypes.RIC**.
msgKey.name	Optional. A **`msgKey.flags`** value of **MsgKeyFlags.HAS_NAME** should be specified, which should match the requested name.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The payload contains a **`Map`** where each entry represents an item in the list. Each map entry contains a **`FieldList`** or **`ElementList`** with additional info about that item.

Table 77: Symbol List Refresh Message Member Use

11.2.3 Symbol List Update Message

A Symbol List Update Message is sent by Open Message Model provider and non-interactive provider applications. It adds or removes items from the list.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.UPDATE == 4**
domainType	**Required**. **DomainTypes.SYMBOL_LIST == 10**
qos	Not used.
updateType	Not used.
seqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
conflationCount	Not used.
conflationTime	Not used.
permData	Optional. Specifies the permission information associated with only the contents of this update.
extendedHeader	Not used.
msgKey.serviceId	**Conditional**. **`msgKey.serviceId`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the ID of the service that provides the item.
msgKey.nameType	**Conditional**. **`msgKey.nameType`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Set this to match the **`msgKey.nameType`** in the item's request message (typically **InstrumentNameTypes.RIC**). If absent, it is assumed to be **InstrumentNameTypes.RIC**.
msgKey.name	**Conditional**. **`msgKey.name`** is required if **RequestMsgFlags.MSG_KEY_IN_UPDATES** was set on the request. Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	**Required**. The payload contains a **`Map`**, where each entry represents an item in the list. Each map entry contains a **`FieldList`** or **`ElementList`** with additional information about that item.

Table 78: Symbol List Update Message Member Use

11.2.4 Symbol List Status Message

A Symbol List status message is encoded and sent by Open Message Model interactive provider and non-interactive provider applications. This message conveys state change information associated with an item stream.

COMPONENT	DESCRIPTION / VALUE
MsgClass	**Required**. **MsgClasses.STATUS == 3**
domainType	**Required**. **DomainTypes.SYMBOL_LIST == 10**
state	Optional. Current state information associated with the data and stream.
qos	Not used.
groupId	Optional. The provider can use this to change the item's **`groupId`**.
permData	Optional. Specifies new permissioning information associated with the stream's contents.
extendedHeader	Not used.
msgKey.serviceId	Optional. Specifies the ID of the service that provides the item.
msgKey.nameType	Optional. **`msgKey.nameType`** should match the name type specified on the request. If it is not specified, **`msgKey.nameType`** defaults to **InstrumentNameTypes.RIC**.
msgKey.name	Optional. Specifies the name of the item being provided.
msgKey.filter	Not used.
msgKey.identifier	Not used.
msgKey.attrib	Not used.
Payload	Not used.

Table 79: Symbol List Status Message Member Use

11.3 Data

11.3.1 Symbol List Request Payload

A consumer requesting a Symbol List stream may also request additional behaviors from a supporting provider. To do so, the payload of its **`RequestMsg`** must contain an **`ElementList`** list that includes an **`ElementEntry`**.

The entry contains an **`ElementList`**, in which the consumer encodes entries to request the desired behaviors. The consumer should ensure that the provider supports the requested behavior by checking the Login Refresh sent by the provider. For more information, refer to Section 3.2.4.

ELEMENT NAME	TYPE	REQUIRED	DESCRIPTION
:SymbolListBehaviors	ElementList	No	Describes the enhanced symbol list behaviors the consumer requests.

Table 80: Symbol List Behaviors Element

11.3.2 Symbol List Data Streams

A Consumer can request that a provider open data streams for the items included in a Symbol List Refresh or Update. Because data streams are opened by the Provider, the Consumer should expect the Provider to use negative **`streamId`** values. To request data streams in this manner, include the following entry in the **`ElementList`** of the **`:SymbolListBehaviors`** entry:

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
:DataStreams	DataTypes.UINT	No	0x0	0x1	Indicates whether the consumer application is requesting that the provider open data streams for items present in the Symbol List. The following flags are defined: • **0x0**: The provider should not open any data streams. • **0x1**: The provider should open streams for items as they are added to the symbol list. • **0x2**: The provider should send non-streaming responses as items are added or updated in the symbol list.

Table 81: Symbol List Behaviors Element

11.3.3 Symbol List Refresh/Update Payload

The Symbol List payload is a **Map**. The provider may include **summaryData** that includes the **domainType** and **nameType** associated with each entry.

Each **MapEntry** key is the item's name. The entry's payload can be empty, contain a **FieldList**, or contain an **ElementList**, either of which can contain additional information (i.e., permission data and cross-reference information). This information should not update frequently.

A **FieldList** typically includes the fields:

- **PROV_SYMB** (3422): Contains the original symbol as provided by the exchange
- **PROD_PERM** (1): Stores permission information

KEY TYPE	CONTAINER TYPE	PERMISSION DATA	REQUIRED	DESCRIPTION
DataTypes.BUFFER DataTypes.ASCII_STRING DataTypes.RMTES_STRING	DataTypes.NO_DATA DataTypes.FIELD_LIST DataTypes.ELEMENT_LIST	Optional	Yes	Contains specific information about an item contained in the list.

Table 82: Symbol List Refresh/Update Map

11.4 Symbol List Sample XML

11.4.1 Symbol List Request Message Sample XML

```
<REQUEST domainType="SYMBOL_LIST" streamId="5" containerType="NO_DATA" flags="0x44
        (STREAMING|HAS_QOS)" Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0 dataSize="0">
    <key flags="0x07 (HAS_SERVICE_ID|HAS_NAME|HAS_NAME_TYPE)" serviceId="1" name="0#ITEMS"
            nameType="1"/>
    <dataBody>
    </dataBody>
</REQUEST>
```

Figure 26. Symbol List Request Message Sample XML Message Layout

11.4.2 Symbol List Refresh Message Sample XML

```
<REFRESH domainType="SYMBOL_LIST" streamId="5" containerType="MAP" flags="0x1E8
        (HAS_MSG_KEY|SOLICITED|REFRESH_COMPLETE|HAS_QOS|CLEAR_CACHE)"  groupId="0"
        Qos: Realtime/TickByTick/Static - timeInfo: 0 - rateInfo: 0 State: Open/Ok/None - text: "Item
        Refresh Completed" dataSize="25">
    <key flags="0x03 (HAS_SERVICE_ID|HAS_NAME)" serviceId="1" name="0#ITEM"/>
    <dataBody>
        <map flags="0x00" countHint="0" keyPrimitiveType="BUFFER" containerType="NO_DATA" >
            <mapEntry flags="0x00" action="ADD" key="N2_UBMS" >
            </mapEntry>
```

```
            <mapEntry flags="0x00" action="ADD" key="TRI.N" >
            </mapEntry>
            <mapEntry flags="0x00" action="ADD" key=".SPX" >
            </mapEntry>
        </map>
    </dataBody>
</REFRESH>
```

Figure 27. Symbol List Refresh Message Sample XML Message Layout

Appendix A UpdateEventTypes

UPDATE EVENT TYPE	VALUE	DESCRIPTION
UpdateEventTypes.UNSPECIFIED	0	Event type of this update is not specified.
UpdateEventTypes.QUOTE	1	The update message contains quote information.
UpdateEventTypes.TRADE	2	The update message contains trade information.
UpdateEventTypes.NEWS_ALERT	3	The update message contains an alert for news information.
UpdateEventTypes.VOLUME_ALERT	4	The update message contains an alert for volume information.
UpdateEventTypes.ORDER_INDICATION	5	The update message contains an order indication.
UpdateEventTypes.CLOSING_RUN	6	The update message is a closing run, intended to reinitialize content between trading days or sessions.
UpdateEventTypes.CORRECTION	7	The update message contains a correction to previously delivered data.
UpdateEventTypes.MARKET_DIGEST	8	The update message is a market digest.
UpdateEventTypes.QUOTES_TRADE	9	The update message contains both trade and quote information.
UpdateEventTypes.MULTIPLE	10	The update message has multiple update types contained in this event.
UpdateEventTypes.VERIFY	11	The update message is provided for applications to verify content with the system.

Table 83: UpdateEventTypes

Appendix B Revision History

RELEASE	REVISIONS
3.5.1.L1	• Rebranded product names
3.5.0.1	• Added Round Trip Time (RTT) latency details
3.0.2	• Added update event type values and their descriptions as an Appendix. • General grammar / typo fixes. • Changed the range for the **`SupportOptimizedPauseResume`** element. • Corrected some information in Section 4.3.1.
3.0	• Correct cross references referring to ‘Section 0’’ • Add content for Yield Curve domain representation • Indicate that **`msgKey.name`** is optional on requests (though required on initial request) and refresh messages.

Table 84: Document Revision History

Document ID: ETAJ360UMRDM.200
Date of issue: October 2020