Enterprise Message API C++ Edition 3.6.8.L1

REFINITIV DOMAIN MODEL USAGE GUIDE

Document Version: 3.6.8
Date of issue: January 2023
Document ID: EMAC368UMRDM.230

REFINITIV™

Contents

Contents

Contents

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 Message API.

1.2 Audience

This guide is written for software developers who are familiar with the Enterprise Message API and want to develop Enterprise Message 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 Message API C++ Edition Developers Guide* and be familiar with the example applications provided in the Enterprise Message 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 Message API provides structural representations of Open Message Model constructs and manages the Refinitiv Wire Format binary-encoded representation of the Open Message Model. Enterprise Message 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 JSON

As of RTSDK 2.0.1.L1, the Enterprise Message API supports WebSocket protocols including **`rssl.json.v2`** and **`tr_json2`** (though Refinitiv intends to deprecate **`tr_json2`** at some time in the future). For further details on WebSocket domain models, refer to the WebSockets API protocol specification on GitHub at https://github.com/Refinitiv/websocket-api/blob/master/WebsocketAPI_ProtocolSpecification.pdf.

1.6 References

For additional Enterprise Message API documentation, refer to:

- The *Enterprise Message API C++ Edition Developers Guide*
- The *Enterprise Message API C++ Edition Reference Guide*
- The Refinitiv Developer Community

1.7 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 ProductDocumentation@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 ProductDocumentation@refinitiv.com address.

1.8 Conventions

1.8.1 Typographic

The Enterprise Message 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 MISSING VARIABLE: structures, functions, 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:

1.8.2 General Transport API Syntax

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

- Dot-separated notation indicates data available within a hierarchy. Each period can indicate a MISSING VARIABLE: structure, a data member**`Name`**, an entry, or an element name.
- **`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.
- Integer constants are defined in all capital letters with underscores (e.g., **MMT_MARKET_PRICE**, **SERVICE_INFO_ID**). In the Enterprise Message API, they can be found in the **refinitiv::ema::rdm namespace** and in the **Access/Include/EmaRdm.h** file.
- The names of Enterprise Message API **`FilterId`** values (e.g. **SERVICE_INFO_ID**) correspond to the flag value enumeration defined for use with the message key's **`filter`** (e.g., **SERVICE_INFO_FILTER**). Names may be shortened for clarity (e.g., **DirectoryInfo**).
- The names of the data members correspond to the method names for both get/set in the Enterprise Message API interface, with the get prefixes removed and the first character always upper case.

1.8.3 Definitions and Standard Behaviors

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

- ***Not Used*** means the attribute is not extensible; the Enterprise Message 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.
- ***Recommended*** means the data is not strictly required, but should be provided or set by all applications.
- ***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.
- ***Extensible*** means the numeric ranges may have more values defined in the future. It means additional Elements can be added to Element Lists.
- If data is not present, the Enterprise Message 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 Message API C++ Edition Developers Guide*.

1.9 Acronyms and Abbreviations

ACRONYM	DEFINITION
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
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
TS1	Time Series One

Table 1: Acronyms and Abbreviations

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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Chapter 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 Message API. Enterprise Message 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 Message Concepts

The following table describes the mapping of Open Message Model concepts with actual interfaces. For clarity and consistency, the Message concept will be referenced throughout the rest of this *Refinitiv Domain Model Usage Guide*.

MESSAGE CONCEPT	DESCRIPTION/VALUE
Request Message	**`ReqMsg`** whose data type is **`DataType.ReqMsgEnum`**
Close Message (Request)	**`OmmConsumer.unregister()`**
Refresh Message (Response)	**`RefreshMsg`** whose data type is **`DataType.RefreshMsgEnum`**
Update Message (Response)	**`UpdateMsg`** whose data type is **`DataType.UpdateMsgEnum`**
Status message (Response)	**`StatusMsg`** whose data type is **`DataType.StatusMsgEnum`**
Post Message	**`PostMsg`** whose data type is **`DataType.PostMsgEnum`**
Generic Message	**`GenericMsg`** whose data type is **`DataType.GenericMsgEnum`**
Ack Message	**`AckMsg`** whose data type is **`DataType.AckMsgEnum`**

Table 3: Message Concepts

2.4 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).

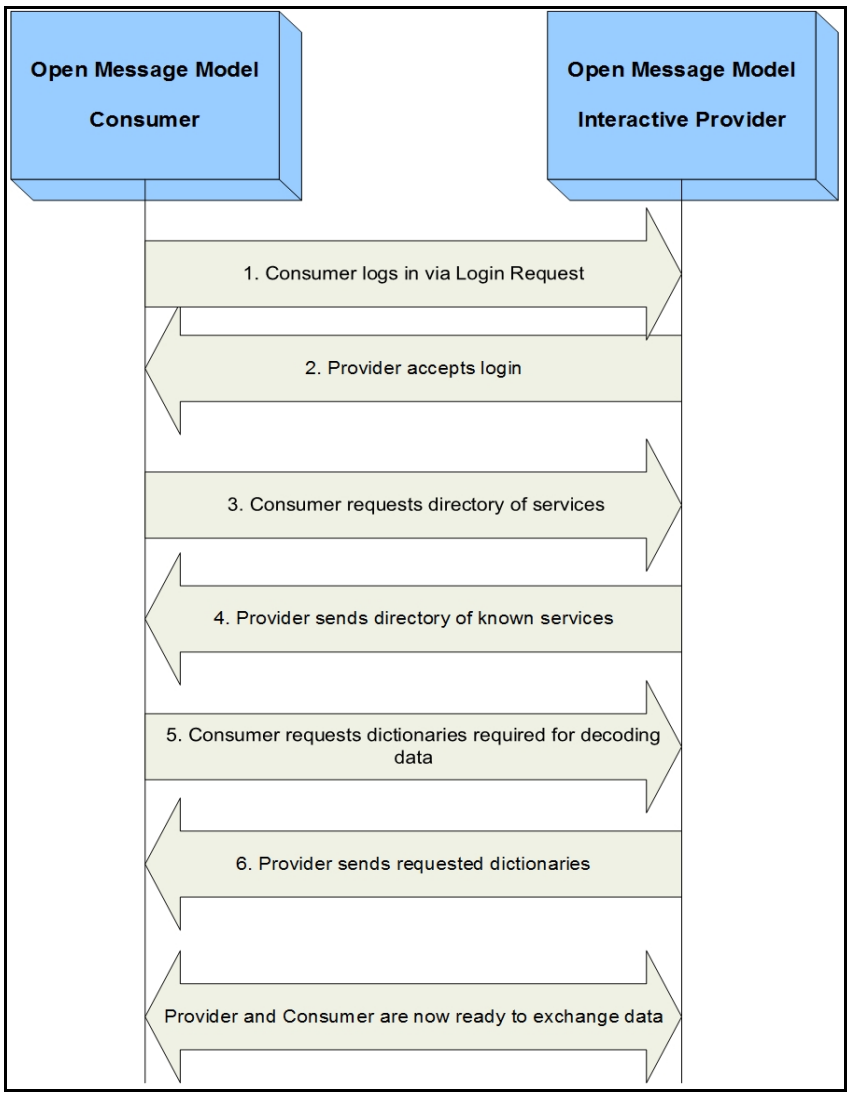


Figure 1. Open Message Model Consumer and Interactive Provider Initial Interactions

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

2.5 Sending and Receiving Content

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

- The consumer sends an **`ReqMsg`** 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`**).

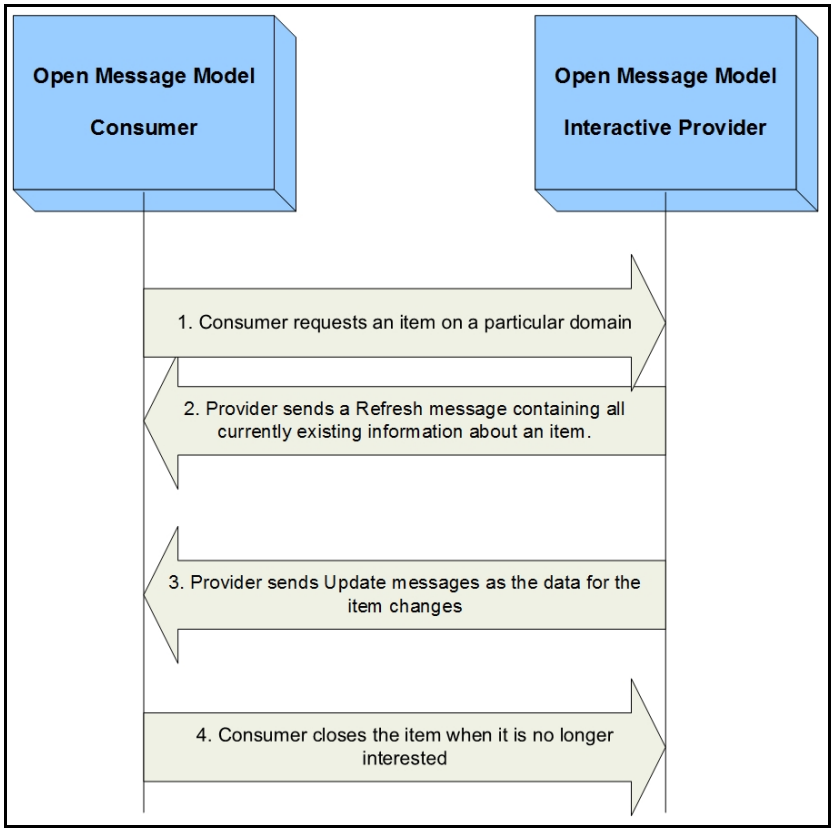


Figure 2. General Domain Use

2.6 General Enterprise Message 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 Message API C++ 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 **ReqMsg.InterestAfterRefresh** 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 **StreamState::NonStreamingEnum**. When the final part of the refresh is received, the stream is considered closed (the final refresh is indicated by the **RefreshMsg.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).

2.6.2 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 **`ReqMsg`** 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 **ReqMsg.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 **ReqMsg.InterestAfterRefresh** 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 Message API C++ Edition Developers Guide*.

2.6.3 Clearing the Cache on Refreshes

If you perform a refresh, you might need to clear the cache. To clear the cache, call **RefreshMsg.ClearCache** with a value of **true**. For further details on using the clear cache flag, refer to the *Enterprise Message API C++ Edition Reference Guide*.

```
RefreshMsg().clearCache(true);
```

When clearing a cache, you must observe the following conditions:

- Pass **true** value on all solicited level 1 data refreshes.
- Pass **true** value only in the first part of solicited level 2 data refreshes.
- Calling this function on unsolicited refreshes depends on the application and its intent:
  - If set to **true** on an unsolicited refresh, the cache is cleared and populated with new data.
  - If not set to **true** on the unsolicited refresh, new data is overlaid onto the existing data. In this case, the resulting image / refresh is a superset of fields currently contained in cache combined with the set brought by the current refresh.

2.6.4 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 Message API C++ Edition Developers Guide*.

NOTE: Currently, the Refinitiv Real-Time Advanced Distribution Server supports view-only on Market Price Level 1 data. As a result, this causes the Enterprise Message API to provide view on Market Price Level 1.

2.6.5 Batch Request

A ***batch request*** allows a consumer application to indicate interest in multiple like-item streams with a single **ReqMsg**. 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 Message API C++ Edition Developers Guide*.

2.6.6 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 Message API C++ Edition Reference Guide*.

3 Login Domain

3.1 Description

The ***Login*** domain registers a user with the system, after which the user can request[1] or post[2] 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.

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 - Section 3.7 cover specific use case scenarios.

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.

3.2 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 internally in the constructor of this class. 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 interactive provider can use the Login request information to authenticate users with the Data Access Control System.

You can configure a login request message using the following methods.

METHOD NAME	DESCRIPTION
OmmConsumerConfig.username()	**Required**. Specifies the user name for login request message.
OmmConsumerConfig.password()	Optional Specifies the password for login request message.
OmmConsumerConfig.position()	Optional Specifies the position for login request message.
OmmConsumerConfig.applicationId()	Optional Specifies the authorization application identifier for login request message.
OmmConsumerConfig.addAdminMsg()	Optional Specifies a login request message to override the default login request.

Table 4: Configure Login Request Message

An initial Login request must be streaming (i.e., a **ReqMsg.InterestAfterRefresh** flag set to **true** is required). After the initial Login stream is established, subsequent Login requests using the same login handle 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 **ReqMsg.Pause** to **true**. To resume data flow on all item streams (also known as a Resume All), users need to call **ReqMsg.InterestAfterRefresh** with **true** value. For more information, refer to the *Enterprise Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_LOGIN = 1**
Interactions	**Conditional**. • Setting **`InitialImage`** to **true** indicates that an initial image is required. • Setting **`InterestAfterRefresh`** to **true** indicates that a streaming request is required. A streaming request is required before any other requests. Non-streaming requests are unsupported. • Setting **`Pause`** set to **true** indicates that a pause is required. A pause request is a request to pause all item streams associated with the login. To resume all item streams associated with the login, issue another streaming request.
QoS	Not used.
worstQos	Not used.
extendedHeader	Not used.
ServiceId	Not used.

Table 5: Login Request Message

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. Possible values are: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** • **USER_COOKIE** • **USER_AUTHN_TOKEN** If **`NameType`** is not set, it is assumed to be **USER_NAME**. A type of **USER_NAME** typically corresponds to a Data Access Control System user name. This can be used to authenticate and permission a user.**USER_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 **`Name`**. To validate users, a provider can pass this user token to an authentication manager application. If you specify **USER_AUTHN_TOKEN**, you must also set **`Name`** to a single, null character (i.e., a 0x00 binary), and include an **`AuthenticationToken`** element in the **`Attrib`**. For details on the **`AuthenticationToken`**, refer to Section 3.2.2. Both **USER_TOKEN** and **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 **USER_TOKEN**, refer to the AAA API documentation. • For further details on using **USER_AUTHN_TOKEN**, refer to the *UserAuthn Authentication User Manual*.[a]
Name	**Required**. **`Name`** should be populated with appropriate content corresponding to the **`NameType`** specification.
Filter	Not used.
Identifier	Not used.
Attrib	Optional. Typically an **`ElementList`**. Attributes are specified in Section 3.2.2.
Payload	Not used.

Table 5: Login Request Message (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 **`Attrib`** elements to send additional authentication information and user preferences between components. The **`ReqMsg.Attrib`** **`ReqMsg.Attrib`** is an **`ElementList`**. The predefined elements available on a Login Request are shown in the following table.

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
AllowSuspectData	UInt	0 | 1	• **1**: Indicates that the consumer application allows a **`OmmState.Suspect`** state. **1** is the default setting. • **0**: Indicates that the consumer application prefers any suspect data result in the stream being closed with an **OmmState.ClosedRecover** state. For more information, refer to Section 3.4.5.

Table 6: Login Request `Attrib` Elements

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
ApplicationAuthorizationToken	ASCII	Sequence of single byte characters from the base36 character set ([0-9][A-Z])	Indicates that application behaviors was inspected and approved by Refinitiv. For more information on obtaining an application authorization token, contact your Refinitiv representative.
ApplicationId	ASCII	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	ASCII	Name of application e.g., **Enterprise Message 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	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.
AuthenticationToken	ASCII	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	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.

Table 6: Login Request `Attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
InstanceId	ASCII	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	ASCII	my_password	Sets the password for logging into the system. This information may be required and encrypted in the future.
Position	ASCII	ip addr/hostname 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	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	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**.
Role	UInt	LOGIN_ROLE_CONS = 0, LOGIN_ROLE_PROV = 1	Indicates the role of the application logging onto the system. • An Open Message Model consumer application should specify its role as **LOGIN_ROLE_CONS**. • An Open Message Model non-interactive provider application should specify its role as **LOGIN_ROLE_PROV**. • Enterprise Message API defaults the role element with a value of 1. 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	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.

Table 6: Login Request `Attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/ EXAMPLE	DESCRIPTION
SingleOpen	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 3.4.5. **`SingleOpen`** defaults to **1**.
SupportProviderDictionaryDownload	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 Message API C++ Edition Developers Guide*. **`SupportProviderDictionaryDownload`** defaults to **0**.

Table 6: Login Request `Attrib` Elements (Continued)

3.2.3 Login Request Domain Representation

The Domain Representation of the Login Request Message is an easy-to-use object which can set up and return an encoded Open Message Model Login Request Message without extensive effort. You can find this object in Enterprise Message API's Login package.

```
OmmConsumerConfig ommConsumerConfig;

ommConsumerConfig.operationModel(
OmmConsumerConfig::UserDispatchEnum );

Login::LoginReq loginRequest;

loginRequest.name("user");
loginRequest.applicationId("127");
loginRequest.position("127.0.0.1/net");
loginRequest.allowSuspectData(true);

ommConsumerConfig.addAdminMsg(loginRequest.getMessage());
```

Figure 3. Login Request Domain Representation Code Usage Example

3.2.4 Login Refresh Message

A Login refresh message is encoded using **RefreshMsg** 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 3.2.7.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_LOGIN = 1**
State	Optional. For the Refresh message, when accepting Login: • **StreamState** = OmmState.Open • **DataState** = OmmState.Ok • **StatusCode** = OmmState.None
Solicited	**Required**. Specifies whether the refresh was solicited. • **true**: Indicates that the refresh was solicited. • **false**: Indicates that the refresh was unsolicited.
Indications	• **Required**: **Complete** set to **true**, which indicates the refresh is complete. The content of a Login Refresh message is expected to be atomic and contained in a single part, therefore **RefreshMsg.Complete** must be set to **true**. • Optional: **ClearCache** set to **true**, which indicates to clear the cache.
QoS	Not used.
SeqNum	Not used.
ItemGroup	Not used.
PermissionData	Not used.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Optional. Possible values: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** • • **USER_AUTHN_TOKEN** If **NameType** is not set then it is assumed to be a **NameType** of **USER_NAME**. If present, the value should match the type specified in the Login request.
Name	Optional. **Name** should match the **Name** specified in the Login request and contain appropriate content corresponding to the **NameType** specification.
Filter	Not used.
Identifier	Not used.

Table 7: Login Refresh Message

COMPONENT	DESCRIPTION / VALUE
Attrib	Optional. Typically an **`ElementList`**. Elements are specified in Section 3.2.5.
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 7: Login Refresh Message (Continued)

3.2.5 Login Refresh Elements

The Login **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	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 **OmmState.ClosedRecover** state. For more information, refer to Section 3.4.5.
ApplicationId	ASCII	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	ASCII	name of application e.g., **Enterprise Message API**	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	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	ASCII	User-defined value	Text accompanying and explaining the **AuthenticationErrorCode**.
AuthenticationExtendedResp	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	UInt	User-defined value	Indicates when a new authentication token needs to be reissued (in UNIX epoch time).

Table 8: Login Refresh Attrib Elements

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
Position	ASCII	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	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	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	UInt	2	Indicates support for **`RoundTripLatency`** monitoring by the provider. If the element is missing, the provider might still support the feature.
SingleOpen	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 3.4.5.
SupportBatchRequests	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 Message API C++ 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.

Table 8: Login Refresh `Attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
SupportEnhancedSymbolList	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.
SupportOMMPost	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 Message API C++ Edition Developers Guide*.
SupportOptimizedPauseResume	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 Message API C++ 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.
SupportPauseResume	UInt	0 | 1	Indicates whether the server supports pause and resume. • **1**: The server supports pause and resume. • **0**: (or if absent): The server does not support pause and resume. **0** is the default setting.
SupportProviderDictionaryDownload	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 Message API C++ Edition Developers Guide*.

Table 8: Login Refresh `Attrib` Elements (Continued)

ELEMENT NAME	DATA TYPE ENUMERATION	RANGE/EXAMPLE	DESCRIPTION
SupportStandby	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 3.2.12.
SupportStandbyMode	UInt	0 | 1 | 2 | 3	Indicates the Warm Standby modes supported by the provider. **`SupportStandby`** needs to be set to 1 in addition to **`SupportStandbyMode`**. • **1**: The provider supports Login-based Warm Standby. • **2**: The provider supports Service-based Warm Standby. • **3**: The provider supports both Login and Service-based Warm Standby. • **0** (or if absent): The provider does not support warm standby functionality. 0 is the default setting.
SupportViewRequests	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 Message API C++ 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.

Table 8: Login Refresh `Attrib` Elements (Continued)

3.2.6 Login Refresh Domain Representation

The Domain Representation of the Login Refresh Message is an easy-to-use object which can set up and return an encoded Open Message Model Login Refresh Message without extensive effort. You can find this object in Enterprise Message API's Login package.

```
Login::LoginRefresh loginRefresh = Login::LoginRefresh();

loginRefresh.allowSuspectData(true);
loginRefresh.singleOpen(true);
loginRefresh.name("user");
loginRefresh.solicited(true);
loginRefresh.state( OmmState::OpenEnum, OmmState::OkEnum, OmmState::NoneEnum, "Login accepted" );

ommProvider.submit( loginRefresh.getMessage(), handle);
```

Figure 4. Login Refresh Domain Representation Code Usage Example

3.2.7 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
DomainType	**Required**. **MMT_LOGIN = 1**
State	Optional. For Status, when rejecting Login: • **`StreamState`** = **OmmState.Closed** • **`DataState`** = **OmmState.Ok** • **`StatusCode`** = **OmmState.NotAuthorized** For Status, when a user needs to retry a login, for example when the Data Access Control System is not yet connected to a Refinitiv Real-Time Advanced Distribution Server: • **`StreamState`** = **OmmState.ClosedRecover** • **`DataState`** = **OmmState.Suspect** • **`StatusCode`** = **OmmState.NotAuthorized**
SeqNum	Optional.
ItemGroup	Not used.
PermissionData	Not used.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Optional. Possible values: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** • **USER_COOKIE** If present, **`NameType`** should match the type specified in the Login request. If **`NameType`** is unspecified, it is assumed to be a **`NameType`** of **USER_NAME**.
Name	Optional. **`Name`** should match the one used in the Login request and should contain appropriate content corresponding to the specification.
Filter	Not used.
Identifier	Not used.
Attrib	Optional. Typically an ElementList. For the contents of ElementList, refer to Section 3.2.5.
Payload	Not used.

Table 9: Login Status Message Member Use

3.2.8 Login Status Elements

The Login **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	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	ASCII		Text accompanying and explaining the **AuthenticationErrorCode**.

Table 10: Login Status Attrib Elements

3.2.9 Login Status Domain Representation

The Domain Representation of the Login Status Message is an easy-to-use object which can set up and return an encoded Open Message Model Login Status Message without extensive effort. You can find this object in Enterprise Message API's Login package.

```
Login::LoginStatus loginStatus = Login::LoginStatus();

if (requestMsg.hasNameType())
    loginStatus.nameType(requestMsg.getNameType());

if (requestMsg.hasName())
    loginStatus.name(requestMsg.getName());

loginStatus.state( OmmState::ClosedEnum, OmmState::SuspectEnum, OmmState::NotFoundEnum, "Invalid
        domain" );

ommProvider.submit(loginStatus.getMessage(), handle);
```

Figure 5. Login Status Domain Representation Code Usage Example

3.2.10 Login Update Message

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

3.2.11 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 3.2.7).

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_LOGIN = 1**
extendedHeader	Not used.
Payload	Not used.

Table 11: Login Close Message Member Use

3.2.12 Login Generic Message Use

3.2.12.1 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 **`{getTicks}`** 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 **`{getTicks}`** 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**. **MMT_LOGIN = 1**
PartNum	Not used.
SeqNum	Not used.
secondarySeqNum	Not used.
PermissionData	Not used.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Not used.
Name	Not used.
Filter	Not used.
Identifier	Not used.
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 12: RTT Login Generic Message Member Use

3.2.13 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 **`Name`** and **`ServiceId`** information. For more details on posting, refer to the *Enterprise Message API C++ Edition Developers Guide*.

3.2.14 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 is posting off-stream and asks for acknowledgments. The acknowledgment contains a positive (ACK) or negative (NACK) code. For more details on posting, see the *Enterprise Message API C++ 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.

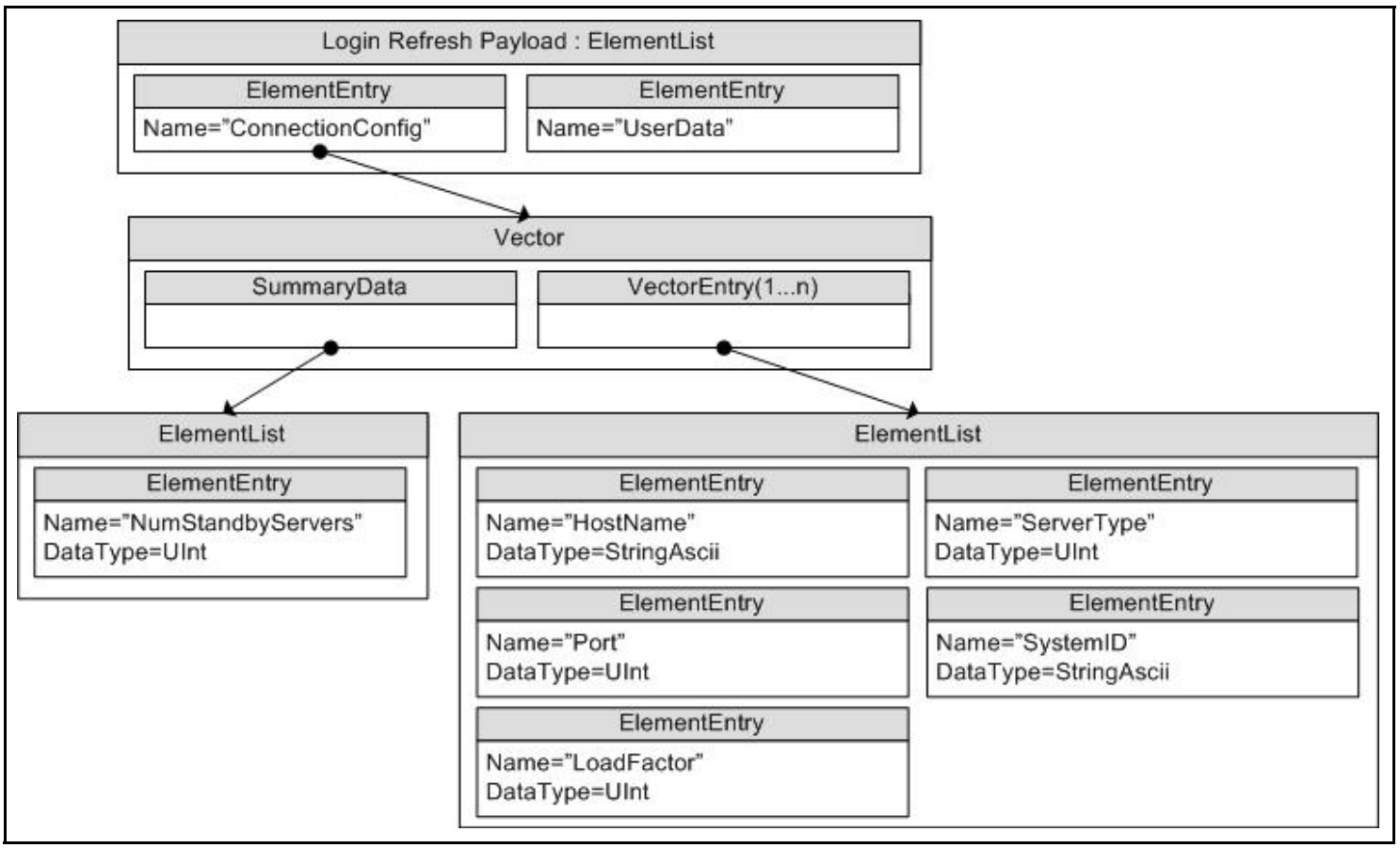


Figure 6. Login Refresh Message Payload

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

NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
NumStandbyServers	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 13: 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	ASCII	"myHostName" "192.168.1.100"	**Conditional**. Specifies the candidate provider's IP address or hostname. **Hostname** is required when a payload is present.
Port	UInt	14002	**Conditional**. Specifies the candidate provider's port number. **Port** is required when a payload is present.
LoadFactor	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	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	ASCII		For future use.

Table 14: ElementList Contents

3.3.2 Login Generic Message Payloads

3.3.2.1 Login Consumer Connection Status Message Payload

The Login data structure for **Payload** is a Map of ASCII -> ElementList. Each key is a **ServiceName**. Each **ElementList** contains one **ElementEntry**.There is no summary data and typically only one map entry that informs the provider of its warm standby role.

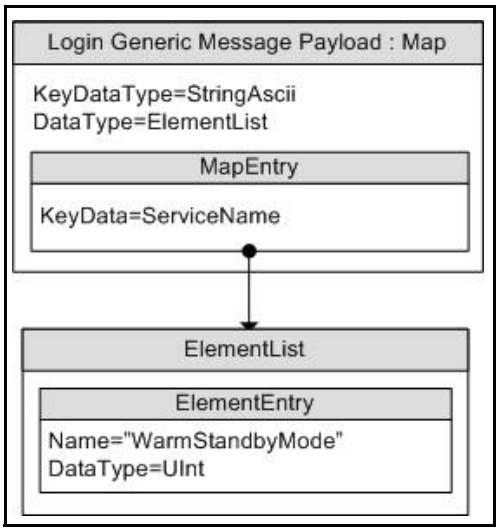


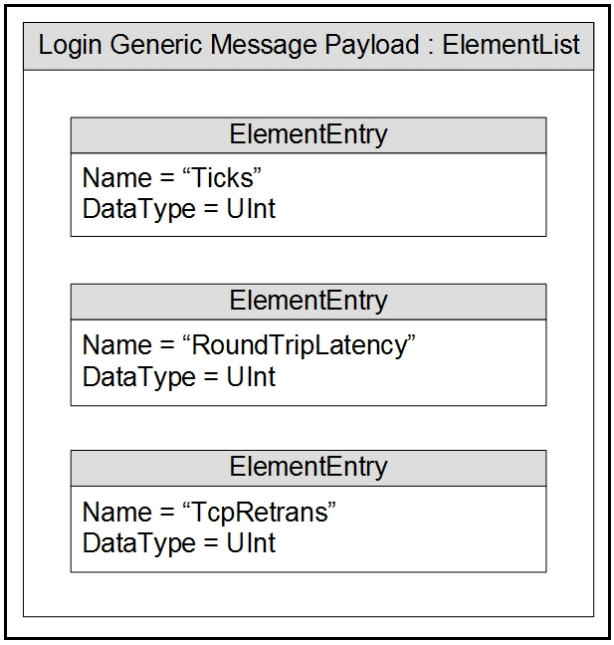
Figure 7. Login Generic Message Payload

ELEMENT NAME	DATA TYPE	RANGE/EXAMPLE	DESCRIPTION
WarmStandbyMode	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: 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	Int	0 – 2^{64} - 1 | 235634	**Required**. Specifies the time set by the provider at the time the message was created.
RoundTripLatency	UInt	0 – 2^{64} - 1 | 358	Specifies the previous Round Trip Latency value (in microseconds) calculated by the provider.
TcpRetrans	UInt	0 – 2^{64} - 1 | 5	Specifies the current number of retransmissions.

Table 16: `ElementList ElementEntrys`

3.4 Special Semantics

3.4.1 Login Direction

Login Request Messages are always sent from client to server, regardless of which is the provider and which is the consumer. Consumers send a Login Request Message to the providers they connect to, while non interactive provider send a Login Request Message to the consumer server.

3.4.2 Initial Login

An Enterprise Message API Consumer sends a login request to an Open Message Model Provider application on behalf of users who are using login attributes specified in **`OmmConsumerConfig`**. Users can register to get a login handle to receive a login status or use the login handle to reissue and post messages on a login stream.

3.4.3 Multiple Logins

Enterprise Message API does not support multiple logins per **`OmmConsumer`** as login stream is opened internally by Enterprise Message API. If multiple logins are needed in the applications, users need to create additional **`OmmConsumer`** instances, one per login.

3.4.4 Group and Service Status

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

3.4.5 Single Open and Allow Suspect Data Behavior

The **SingleOpen** and **AllowSuspectData** Elements that are passed via the **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.

The status in the table could be from a Directory STATE filter entry, from a Directory GROUP filter entry, or from an item Status Message. For more information on Status, refer to the *Enterprise Message API C++ Edition Developers Guide*.

NOTE: The Enterprise Message 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
New item requests when[a]: **ServiceState** = UP or DOWN **AcceptingRequests** = FALSE	**SS** = OPEN **DS** = SUSPECT based on individual item's state	**SS** = OPEN **DS** = SUSPECT based on individual item's state	**SS** = CLOSED_RECOVER **DS** = SUSPECT based on individual item's state	**SS** = CLOSED_RECOVER **DS** = SUSPECT
Connection goes down	**SS** = OPEN **DS** = SUSPECT based on individual item's state	**SS** = OPEN **DS** = SUSPECT based on individual item's state	**SS** = CLOSED_RECOVER **DS** = SUSPECT based on individual item's state	**SS** = CLOSED_RECOVER **DS** = SUSPECT

Table 17: SingleOpen and AllowSuspectData Handling

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

3.5 Specific Usage: RDF Direct Login

When sending a Login Request message to an RDF Direct, the **Name** can be an ASCII string composed of printable characters. The Name is used for scoping some RDF Direct's configuration. The **NameType** must be **USER_NAME**.

In the **Attrib**'s **ElementList**, **SingleOpen**, **AllowSuspectData**, and **ProvidePermissionExpressions** are supported. **ApplicationId**, **Position**, **Password**, and **ProvidePermissionProfile** are ignored.

The request and response message **Payload**s have no data.

RDF Direct supports only one login per connection.

3.6 Specific Usage: RDMS

When sending a Login to a Data Access Control System-enabled Refinitiv Real-Time Distribution System deployment, **Name** should be a valid username in the Data Access Control System. The **Attrib** containing **ApplicationId** and **Position** will also be used for Data Access Control System authorization. **Name** is used for scoping aspects of the Refinitiv Real-Time Distribution System configuration. **NameType** must be **USER_NAME**.

In the **Attrib**'s **ElementList**, **ApplicationId**, **Position**, **SingleOpen**, **AllowSuspectData**, and **ProvidePermissionExpressions** are supported. **Password** and **ProvidePermissionProfile** are ignored. They may be echoed to the consumer, but the values are not necessarily correct.

Request and response message **Payload**s have no data.

Refinitiv Real-Time Distribution System supports only one login per connection.

3.7 Specific Usage: Login Credentials Update Feature

Internally Enterprise Message API stores all login credentials (e.g., user name, name type, login attributes), so it can use them later during connection recovery phase. These credentials can be changed by the application at any point in time after a connection and login is established. To change login credentials, an application needs to reissue a login request message with the new credentials. This new request message must meet the following criteria:

- A new user name parameter, different from the one specified on a prior request or reissue, must be specified.
- The **NameType** parameter must be specified as **USER_TOKEN** on all, the initial request and all subsequent reissues.
- The Interactions set in the reissue must match the original **InteractionType**.

If all of the above conditions/criteria are met, Enterprise Message API will send the reissue with the new login credentials to the server and will internally store the new/updated login credentials to be used later during connection recovery. If all of the above conditions/criteria are not met, Enterprise Message API will apply the standard login reissue processing. If no new/updated login credentials are specified by application, during the connection recovery phase Enterprise Message API will use the previously used ones.

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 **`ServiceName`** or **`ServiceId`**.
- 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 using **ReqMsg** with default or user-configured values and sent internally by **OmmConsumer** in the constructor of this class. A consumer can request information about all services by omitting **ServiceName** or **ServiceId** information, or specify a **ServiceName** or **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 **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 Message API C++ Edition Reference Guide*.

Users can configure a directory request message using the **OmmConsumerConfig.addAdminMsg()** to override the default directory request.

Refinitiv recommends that a consumer application minimally request **SERVICE_INFO_FILTER** and **SERVICE_STATE_FILTER** 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 **ServiceName** or **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).

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DIRECTORY = 4**
Interactions	**Required**. • **InitialImage**: **true**, indicates initial image is required • **InterestAfterRefresh**: **true**, indicates streaming request is required Only streaming and non-streaming requests are supported. If you need to send a ConsumerStatus generic message over the Directory stream, you must specify the interaction as "Streaming" in the request.
QoS	Not used.
worstQos	Not used.
priorityClass	Not used.
priorityCount	Not used.
extendedHeader	Not used.
NameType	Not used.
Name	Not used.
Filter	**Required**. Specifies a filter indicating the specific data in which a consumer is interested. Available categories include: • **SERVICE_INFO_FILTER = 0x01** • **SERVICE_STATE_FILTER = 0x02** • **SERVICE_GROUP_FILTER = 0x04** • **SERVICE_LOAD_FILTER = 0x08** • **SERVICE_DATA_FILTER = 0x10** • **SERVICE_LINK_FILTER = 0x20** For details on the contents of each filter entry, refer to Section 4.3.1.1.

Table 18: Source Directory Request Message

COMPONENT	DESCRIPTION / VALUE
ServiceName	Optional. • If present, the directory request is for the specified service. • If neither the **`ServiceId`** nor the **`ServiceName`** is specified, then the request is for the entire directory.
	NOTE: If the application requests a specific service, it should set either the **`ServiceId`** or **`ServiceName`** of the service, but not both.
ServiceId	Optional. • If present, the directory request is for the specified service. • If neither the **`ServiceId`** nor the **`ServiceName`** is specified, then the request is for the entire directory.
	NOTE: If the application requests a specific service, it should set either the **`ServiceId`** or **`ServiceName`** of the service, but not both.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 18: Source Directory Request Message (Continued)

4.2.2 Source Directory Refresh Message

A Directory Refresh Message is encoded using a **RefreshMsg** and 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
DomainType	**Required**. **MMT_DIRECTORY = 4**
State	**Required**. Indicates stream and data state information.
Solicited	**Required**. Specifies whether the refresh was solicited. Available values are: • **true**: Indicates the refresh was solicited. • **false**: Indicates the refresh was unsolicited.
Indications	**Conditional**. • **Complete**: **true**, indicates refresh complete • **ClearCache**: **true**, indicates clear cache • **DoNotCache**: **true**, indicates this refresh message must not be cached. For more details, refer to the FilterEntries in Section 4.3.1.
QoS	Not used.
SeqNum	Optional. A user-specified, item-level sequence number that the application can use to sequence messages within this stream.
ItemGroup	Not used.
PermissionData	Not used.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Not used.
Name	Not used.
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 **Filter** member in Section 4.2.1.
Identifier	Not used.
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 **ServiceName**. For additional details, refer to Section 4.3.1.

Table 19: Source Directory Refresh Message

4.2.3 Source Directory Update Message

A Source Directory Update Message is encoded using an **`UpdateMsg`** and 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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DIRECTORY = 4**
Indications	**Conditional**. • **`DoNotCache`**: **true**, indicates this update message must not be cached. • **`DoNotConflate`**: **true**, indicates this update message must not be conflated. For more details, refer to the FilterEntries in Section 4.3.1.
UpdateTypeNum	Not used.
SeqNum	Optional. A user-specified, item-level sequence number that the application can use to sequence messages in this stream.
ConflatedCount	Not used.
ConflatedTime	Not used.
PermissionData	Not used.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Not used.
Name	Not used.
Filter	Optional. The **`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 **`Filter`** member in Section 4.2.1.
Identifier	Not used.
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

4.2.4 Source Directory Status Message

A Source Directory status message is encoded using a **`StatusMsg`** 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. Such state information can indicate that a directory stream cannot be established or to inform a consumer of a state change associated with an open directory stream. The Directory Status message can also be used to close an existing directory stream.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DIRECTORY = 4**
State	Optional. Contains stream and data state information for the directory stream. • StreamState • DataState • StatusCode
ItemGroup	Not used.
Indications	Optional. **`ClearCache`**: **true**, Indicates the application should clear its cache For more details, refer to the FilterEntries described in Section 4.3.1.
PermissionData	Optional. If present, this is the new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Not used.
Name	Not used.
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 **`Filter`** member in
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 21: Source Directory Status Message

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
DomainType	**Required**. **MMT_DIRECTORY = 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.
PermissionData	Not used.
extendedHeader	Not used.
NameType	Not used.
Name	**Required**. The name of this message must be **`ConsumerStatus`**.
Filter	Not used.
Identifier	Not used.
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

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 **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 **Filter** member of the refresh or update. These categories are described in Table 24.

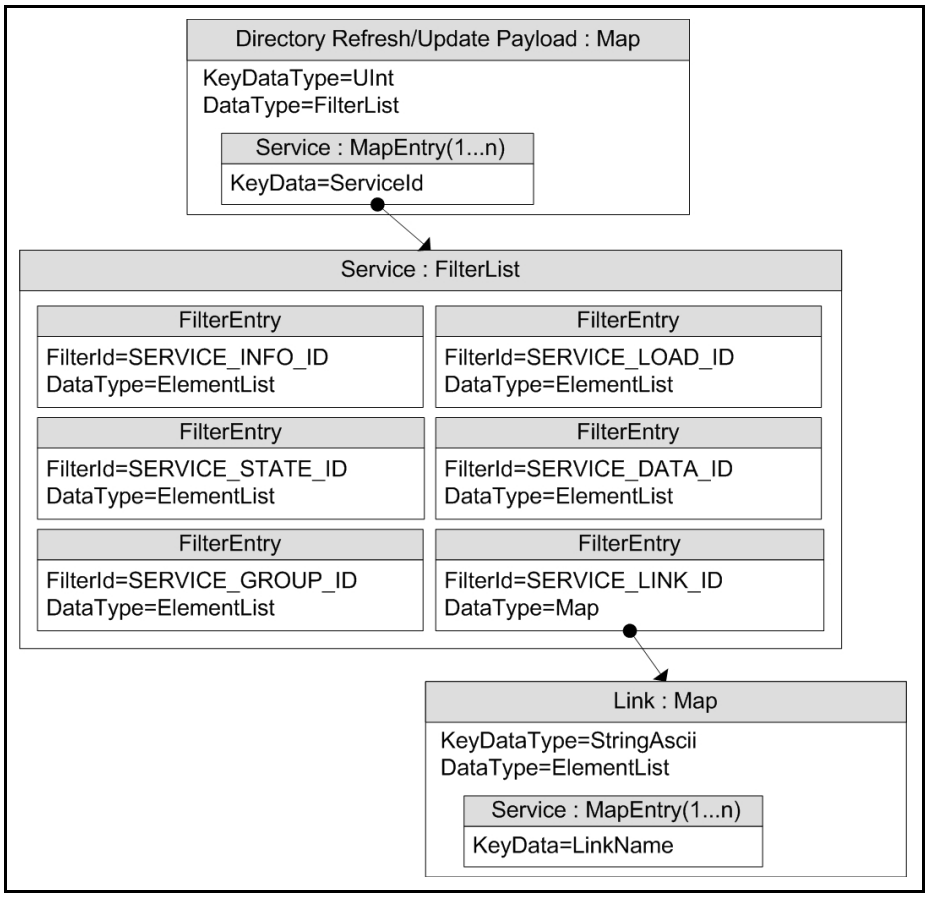


Figure 8. Source Directory Refresh and Update Message Payload

KEY TYPE	CONTAINER TYPE	PERMISSION DATA	DESCRIPTION
UInt for a service ID	FilterList	Not used	Contains information for each known service. The key is the service's **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 **FilterEntry.Clear** is not used, and that the Info category should not change) for Directory and Dictionary domain message models as part of a reissue. None of these categories use permission data. In the following table, the description for each **FilterEntry** includes whether the content is extensible.

FILTERENTRY ID (CORRESPONDING FILTER BIT-VALUE)	TYPE	DESCRIPTION
SERVICE_INFO_ID= (SERVICE_INFO_FILTER=)	ElementList	**Provider applications must be able to provide this information.** Identifies a service and its available data. This content is extensible. Refer to.
SERVICE_STATE_ID (SERVICE_STATE_FILTER=	ElementList	**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. This content is not extensible. The effects of this category occur immediately. Therefore, the initiating **`UpdateMsg`** should set **DoNotConflate** to **true**. Refer to.
SERVICE_GROUP_ID= (SERVICE_GROUP_FILTER=)	ElementList	Manages group information. Can change the status of a group of items or merge items from one group to another. This content is not extensible. The effects of this category occur immediately and only affect existing items. Therefore, the initiating **`UpdateMsg`** should set **DoNotConflate** and **DoNotCache** to **true**. Refer to.
SERVICE_LOAD_ID= (SERVICE_LOAD_FILTER=)	ElementList	Information about the current allowable workload of this service, including how many items are currently being serviced. This content is extensible. Optionally, the initiating **`UpdateMsg`** can set **DoNotConflate** to **true**. Refer to.
SERVICE_DATA_ID= (SERVICE_DATA_FILTER=)	ElementList	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. This content is extensible. Flag values **DoNotConflate** and **DoNotCache** can optionally be set to **true** to prevent conflation and caching of this content. Refer to.
SERVICE_LINK_ID (SERVICE_LINK_FILTER)	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. This content is not extensible. Refer to.

Table 24: Source Directory `MapEntry` Filter Entries

4.3.1.1 Source Directory Info Filter Entry

The Info filter entry (**SERVICE_INFO_FILTER**, **SERVICE_INFO_ID**) 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	ASCII	e.g., IDN_RDF	**Required.** Specifies the service's name. This will match the concrete service name or the service group name that is in the **`Map.Key`**.
Vendor	ASCII	e.g., Refinitiv	Specifies the name of the vendor that provides the data for this service.
IsSource	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
Capabilities	Array of UInt	e.g., [5, 6]	**Required.** Lists the domains which this service can provide. Note that the UInt MesageModelType is extensible, using values defined in this guide (i.e., 1-255). For example, a list containing **MMT_DICTIONARY** (**5**) and **MMT_MARKET_PRICE** (**6**) indicates a consumer can request dictionaries and Market Price data from this service.
DictionariesProvided	Array of ASCII	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 **MMT_DICTIONARY** domain. For details, refer to Chapter 5, Dictionary Domain.
DictionariesUsed	Array of ASCII	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 Chapter 5, Dictionary Domain.

Table 25: Source Directory Info Filter Entry Elements

ELEMENT NAME	TYPE	RANGE/ EXAMPLE	DESCRIPTION
QoS	Array of QoS	e.g., Real-time, TickByTick	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, 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 Message API C++ Edition Reference Guide*.
SupportsQoSRange	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 **`ReqMsg.Qos`**. • **0**: The provider does not support QoS range requests. This is the default behavior. • **1**: The provider supports QoS range requests.
ItemList	ASCII		Specifies the name of a SymbolList (i.e., a specific item requested to get the names of all items available for this service). If it is not present, this feature is not supported. The consumer requests this item via the **MMT_SYMBOL_LIST** domain (See Chapter 11, Symbol List Domain).
SupportsOutOfBandSnapshots	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	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 (**SERVICE_STATE_FILTER**, **SERVICE_STATE_ID**) 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 **OmmState.ClosedRecover**.

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

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
ServiceState	UInt	0 | 1	**Required.** Indicates whether the original provider of the data is available to respond to new requests. Changes to **ServiceState** do not affect streams that are already open. Available values are: • **0**: Service is Down • **1**: Service is Up Refer toSection 4.4.3 .
AcceptingRequests	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 toSection 4.4.3 .
Status	State	e.g., OmmState.Open, OmmState.Ok, OmmState.None, "OK"	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 **OmmState.Open** or **OmmState.ClosedRecover**. This status only applies to item streams that have received a refresh or status of **OPEN**/**OK**. Refer to Section 4.4.4.1.

Table 26: Source Directory State `FilterEntry` Elements

4.3.1.3 Source Directory Group Filter Entry

The Group filter entry (**SERVICE_GROUP_FILTER**, **SERVICE_GROUP_ID**) conveys item group status and item group merge information. Every item stream is associated with an item group as defined by the **`ItemGroup`** 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 Message API C++ 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	Buffer	e.g., 1.26.102	**Required**. Specifies the **`ItemGroup`** 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	Buffer	e.g., 1.26.110	Changes all items whose group currently matches the **Group** element to the specified **`MergedToGroup`**.
Status	State	e.g., StreamState::OpenEnum, DataState::OkEnum, StatusCode::NoneEnum, "OK"	A status change to be applied to all items whose **`ItemGroup`** matches the **Group** element. It is equivalent to sending a **`StatusMsg`** to each item. • The **`streamState`** is only allowed to be **OmmState.Open** or **OmmState.ClosedRecover**. • If you need to convey group status **`text`** or **`code`** information without changing the data state, use the value **DataState::NoChangeEnum**. • 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 4.4.4.2

Table 27: Source Directory Group `FilterEntry` Elements

4.3.1.4 Source Directory Load Filter Entry

The Load filter entry (**SERVICE_LOAD_FILTER**, **SERVICE_LOAD_ID**) 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	UInt	0 – MAXUINT	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	UInt	0 - MAXUINT	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	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 (**SERVICE_DATA_FILTER**, **SERVICE_DATA_ID**) 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	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 (**SERVICE_LINK_FILTER**, **SERVICE_LINK_ID**) 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 **ASCII**. 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 typical consumer application can treat this entry as mainly informational. The consumer should use the **`State`** category 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	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	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	UInt	0 - 3	Provides additional information about the upstream source. • **0**: None (this is the default) • **1**: Ok • **2**: RecoveryStarted • **3**: RecoveryCompleted
Text	ASCII	N/A	Explains the **`LinkState`** and **`LinkCode`**. Text defaults to "".

Table 30: Source Directory Link `FilterEntry` Map 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 directory data structure for the ConsumerStatus message is a **`Map`**. Each **`MapEntry`** sends status to one service and is uniquely identified by **`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.

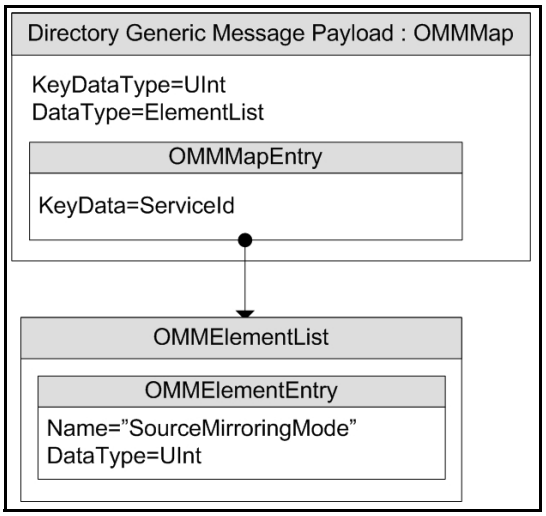


Figure 9. Source Directory Generic Message Payload

ELEMENT NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
SourceMirroringMode	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 31: 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 Service IDs

Most Refinitiv Domain Model messages can be associated with a service (although Login and Directory typically are not). For better bandwidth utilization, the RSSL transport optimizes the service name into a two byte service ID. The **`ServiceId`** is only unique within a single channel.

4.4.3 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 values of **ServiceState** and **AcceptingRequests** do not affect existing streams and do not imply anything about the data quality of existing streams.

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. 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 32: ServiceState and AcceptingRequests

4.4.4 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.

4.4.4.1 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 (for details, refer to Section 2.3) 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.

4.4.4.2 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.

4.4.5 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 **MapEntry.Delete**. A consumer should place all open items associated with this service in the **OmmState.ClosedRecover**.

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 **OmmState.Closed** or **OmmState.ClosedRecover** on the Source Directory stream.
- The provider sends a message with a **ClearCache** on a **`StatusMsg`** on the Source Directory stream.

If any of these events occurs, all of the items for the service(s) are automatically cleaned up and considered to have a ClosedRecover status.

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.

4.4.6 Automatic Request from Enterprise Message API Consumer

Enterprise Message API internal consumer implementation will always automatically request a Directory with the **`Filter`** set to **`SERVICE_INFO_FILTER`**, **`SERVICE_STATE_FILTER`**, or **`SERVICE_GROUP_FILTER`**. This ensures that Enterprise Message API can:

- Map service IDs to names
- Fanout SERVICE_STATE_ID.Status and SERVICE_GROUP_ID.Status
- Apply SERVICE_GROUP_ID.MergedToGroup

The Directory request is sent after the Login is successful. Response message for this directory request are not forwarded to the consumer application. If the consumer wants source directory information, it is required to make its own request for the Directory.

4.4.7 Client Requests Non-Existing Service Directory

If the client sends a directory request without specifying service name or service ID, the directory response includes all available services. If the client specifies a service name or service ID in a directory request, it receives the directory response for just the requested service. If the requested service name or service ID is not available, Enterprise Message API should send a service directory containing an empty map entry in the payload. If the service becomes available later, the client receives an update message which contains the required service information.

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., **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**. 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.

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 **Data**. Because the Enterprise Message API internally stores pre-decoded data, an application can easily decode a **FieldEntry** (without cross-referencing the **FieldId** to the correct Field Dictionary to determine its type).

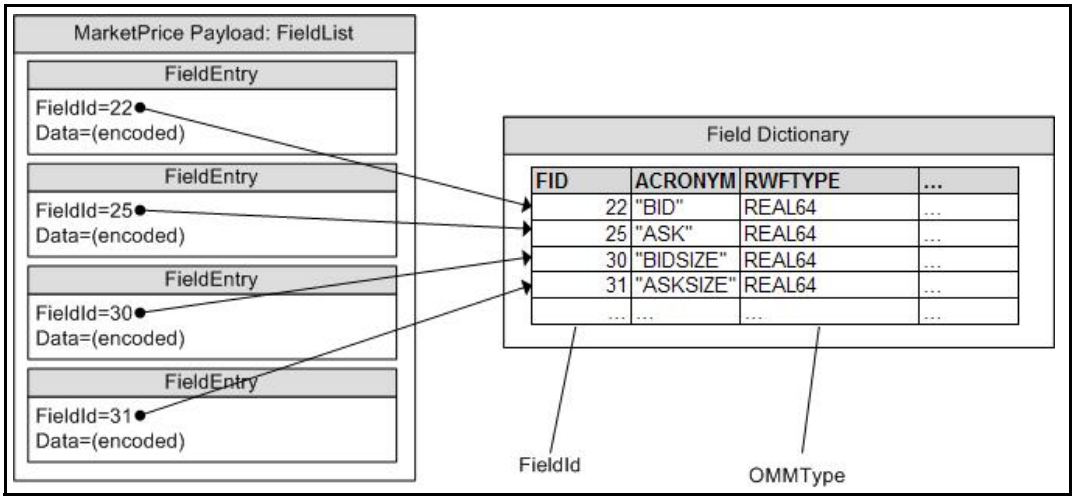


Figure 10. FieldList Referencing Field Dictionary

If the field's type is **DataTypeEnum::EnumEnum**, there may be a table of values in the corresponding Enumerated Types Dictionary. The consumer can then reference that information.

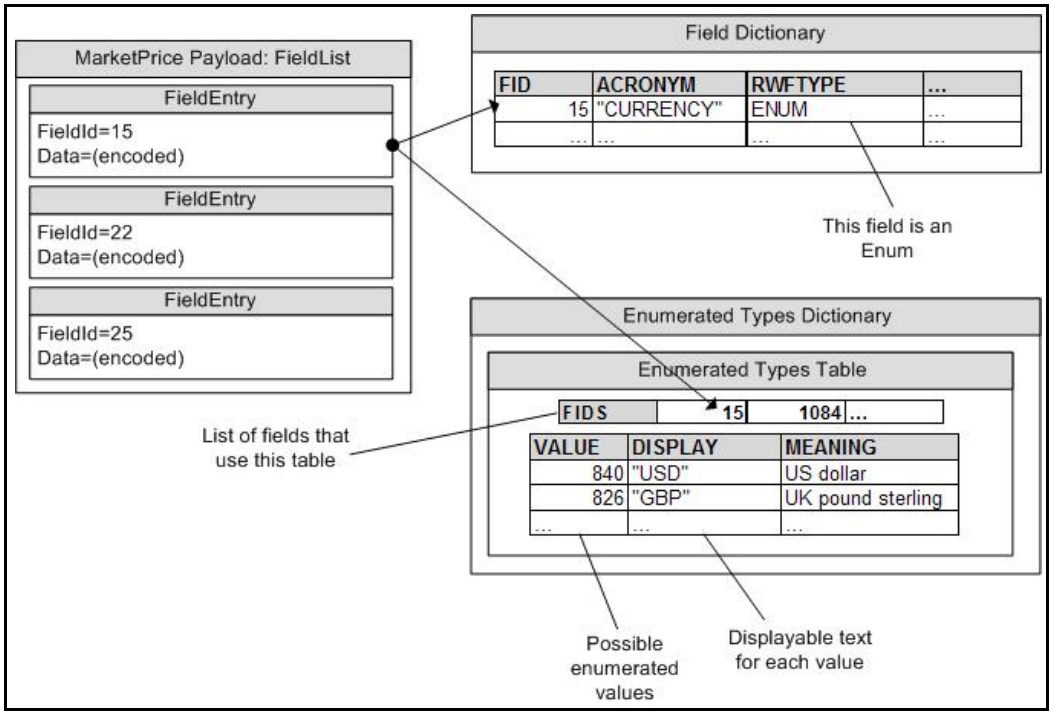


Figure 11. FieldEntry Referencing an Enumerated Types Table

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 using **ReqMsg** and sent internally by the **OmmConsumer** in the constructor of this class. The request indicates the name of the desired dictionary and how much information from that dictionary is needed.

Users can configure dictionary request messages using **OmmConsumerConfig.addAdminMsg()** to override the default dictionary request.

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

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DICTIONARY = 5**
Interactions	**Required**. • **InitialImage**: **true**, Indicates an initial image is required. • **InterestAfterRefresh**: **true**, Indicates a streaming request is required. After receiving **RefreshComplete**, the consumer can only receive a ***Status response*** message. An ***Update response*** message will never be received. Pause request is not supported.
QoS	Not used.
worstQos	Not used.
priorityClass	Not used.
priorityCount	Not used.
Priority	Optional.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the **ServiceName** of the service from which the consumer requests the dictionary.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the **ServiceId** of the service from which the consumer requests the dictionary.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Not used.
Name	**Required**. Specifies the **Name** of the desired dictionary as seen in the Source Directory response (refer to Section 4.3.1.1).
Filter	**Required**. The filter represents the desired verbosity of the dictionary. The consumer should set the **Filter** according to how much information is needed: • **DICTIONARY_INFO = 0x00**: Provides version information only. • **DICTIONARY_MINIMAL = 0x03**: Provides information needed for caching. • **DICTIONARY_NORMAL = 0x07**: Provides all information needed for decoding. • **DICTIONARY_VERBOSE = 0x0F**: Provides all information (including comments). Providers are not required to support the **MINIMAL** and **VERBOSE** filters. For further details on **Filter**, refer to Section 5.4.1.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 33: Dictionary Request Message

5.3.2 Dictionary Refresh Message

A Dictionary refresh message is encoded and sent by OMM Interactive and non-interactive provider applications and provides the consumer with the content of the requested dictionary. A dictionary refresh may be encoded in one or multiple parts.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DICTIONARY = 5**
Indications	Optional: • **`DoNotCache`**: **true**, Indicates the application should not cache • **`ClearCache`**: **true**, Indicates the application should clear the cache • **`Complete`**: **true**, Indicates a refresh complete
state	**Required**. Indicates stream and data state information.
QoS	Not used.
seqnum	Optional. A user-specified sequence number that the application can use for sequencing messages within this stream.
groupId	Not used.
PermissionData	**Conditional**. Used if the provided dictionary requires permissioning.
extendedHeader	Not used.
ServiceId	**Required**. Specifies the **`ServiceId`** of the service that provides the dictionary. **NOTE:** The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
ServiceName	**Required**. Specifies the name of the service that provides the dictionary. **NOTE:** The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
NameType	Not used.
Name	**Required**. Specifies the name of the provided dictionary, as advertised as supported in the Source Directory response (refer to Section 4.3.1.1).
Filter	**Required**. The filter represents verbosity of dictionary in the response message. When possible, this should match the filter set in the consumer's request. For additional details, refer to the **`Filter`** member in Section 5.3.1.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. The payload structure varies depending on the dictionary's type. However, the payload is typically a **`Series`** containing an **`ElementList`**, while the series **`SummaryData`** indicates the specific dictionary type.

Table 34: Dictionary Refresh Message

5.3.3 Dictionary Status Message

A dictionary status message is encoded using **`StatusMsg`** 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
DomainType	**Required**. **MMT_DICTIONARY = 5**
State	Optional. Contains stream and data state information for the dictionary stream.
ItemGroup	Not used.
Indications	Optional: • **`ClearCache`**: **true**, Indicates the application should clear the cache
PermissionData	**Conditional**. Used if the provided dictionary requires permissioning.
extendedHeader	Not used.
ServiceId	Not used.
NameType	Not used.
Name	Not used.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 35: Dictionary Status Message

5.4 Data

5.4.1 Filter

While Dictionary's Filters values correlate to a bitmap, the Enterprise Message API supports only the combinations in the following table. For example, a Dictionary's **Filter** cannot be **0x2** or **0x6**.

Dictionary providers are required to support **DICTIONARY_INFO** and **DICTIONARY_NORMAL** filers. **Filter** can be changed for Directory and Dictionary domain message models as part of a reissue. It cannot be changed in other domain message models. If an unsupported **Filter** is requested, the provider may do either of the following:

- Change the **Filter** in response message to a supported one.
- Send a Closed State in the response message.

MASK	PROVIDER MUST SUPPORT IN RESPONSE MESSAGE	DESCRIPTION
DICTIONARY_INFO=0x0	Yes	Dictionary summary information, such as DictionaryType and version. The response **Payload.SummaryData** will contain data but the response payload will contain no entries.
DICTIONARY_MINIMAL=0x3	No	DICTIONARY_INFO plus the minimum data needed to cache or convert data.
DICTIONARY_NORMAL=0x7	Yes	DICTIONARY_MINIMAL plus all other data, except descriptions and comments.
DICTIONARY_VERBOSE=0xF	No	All available data.

Table 36: Dictionary's Filter

5.4.2 Refresh Message Summary Data

A dictionary's SummaryData is an **ElementList** that can be used by a consumer to find out if it needs an updated dictionary or if it needs the dictionary at all. SummaryData is extensible and can include other elements.

NAME	TYPE	RANGE/EXAMPLE	DESCRIPTION
Version	ASCII	"1.0.1"	**Required**. Specifies the version of the provided dictionary. For additional details on dictionary versions, refer to Section 5.7.2. **NOTE:** The Enumerated Types dictionaries populate the Version element using information from the **DT_Version** tag.
Type	UInt	Total range is from 0 to 255, where values 0 - 127 are reserved and values 28-255 are extensible. • DICTIONARY_FIELD_DEFINITIONS = 1 • DICTIONARY_ENUM_TABLES = 2 • DICTIONARY_RECORD_TEMPLATES = 3 • DICTIONARY_DISPLAY_TEMPLATES = 4 • DICTIONARY_DATA_DEFINITIONS = 5 • DICTIONARY_STYLE_SHEET = 6 • DICTIONARY_REFERENCE = 7	**Required**. Indicates the type of dictionary contained in the payload.
DictionaryId	Int	Total range is from -16383 to 16383, where: • Values 0 to 16383 are reserved by Refinitiv • The value **1** corresponds to the **RDMFieldDictionary**. • The value **0** signifies 'Unspecified' • Values -1 to -16383 are Extensible	Enterprise Message API can use **DictionaryId** in field lists and series to associate fields with field definitions or enumerations. Refer to Section 5.4.4. **DictionaryId** defaults to **0**.
RT_Version	ASCII	"1.0.1"	Optionally sent only with the enumerated type dictionary. **RT_Version** identifies which field dictionary should be used with this enumerated type dictionary.
DT_Version	ASCII	"1.0.1"	Optionally sent only with the enumerated type dictionary. **DT_Version** conveys the display template version.

Table 37: Dictionary summaryData

5.4.3 Response Message Payload

The Response Message (refer to Section 2.3) payload can vary widely, based on its DictionaryType. The payload is typically a Series of ElementLists, but can also be XML or Opaque data. For further details on the response message payloads, refer to Section 5.5.1 and Section 5.6.1.

Some DictionaryTypes also have external file representations of their data. For details about the data of each DictionaryType, refer to Section 5.5.2 and Section 5.6.2.

5.4.4 DictionaryId

The first **FieldList** provided for an item always has a **DictionaryId**. While a **FieldList** can be parsed without a Dictionary, to interpret the data, the **FieldList**'s **DictionaryId** must be associated with a Dictionary. The **DictionaryId** (provided in a Dictionary response message's **Payload.SummaryData**) associates a **FieldList**'s **DictionaryId** to a "family" of Dictionaries.

A Dictionary family includes a single FieldDefinition Dictionary. Enumeration tables for a single FieldDefinition Dictionary must be consolidated into a single EnumTable Dictionary that has the same **DictionaryId** as the FieldDefinition Dictionary. The Dictionary family may also include a single RecordTemplate Dictionary and a single DisplayTemplate Dictionary.

The **DictionaryId** is **0** for StyleSheet and Reference. A **DictionaryId** setting of **0** means unspecified, so the Dictionary is not used for parsing, interpreting, or displaying **FieldList**s. For example, a "TimeZone" reference dictionary may include table information about every world time zone. Because timezone information is not needed to parse **FieldList**s, there is no need to assign a **DictionaryId** to the "TimeZone" Dictionary. Thus, the value of its **DictionaryId** is set to **0** (i.e., unspecified).

DictionaryIds are globally scoped can have the range of -16383 to 16383. Though **DictionaryId**s 0 through 16383 are reserved for use by Refinitiv, applications can provide their own dictionaries by selecting a **DictionaryId** between -1 and -16383. If a single **FieldList** needs to use fields defined in two dictionaries, the **FieldList** can specify a dictionary switch using 0 for the Field ID. For details, refer to the *Enterprise Message API C++ Edition Developers Guide*.

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.

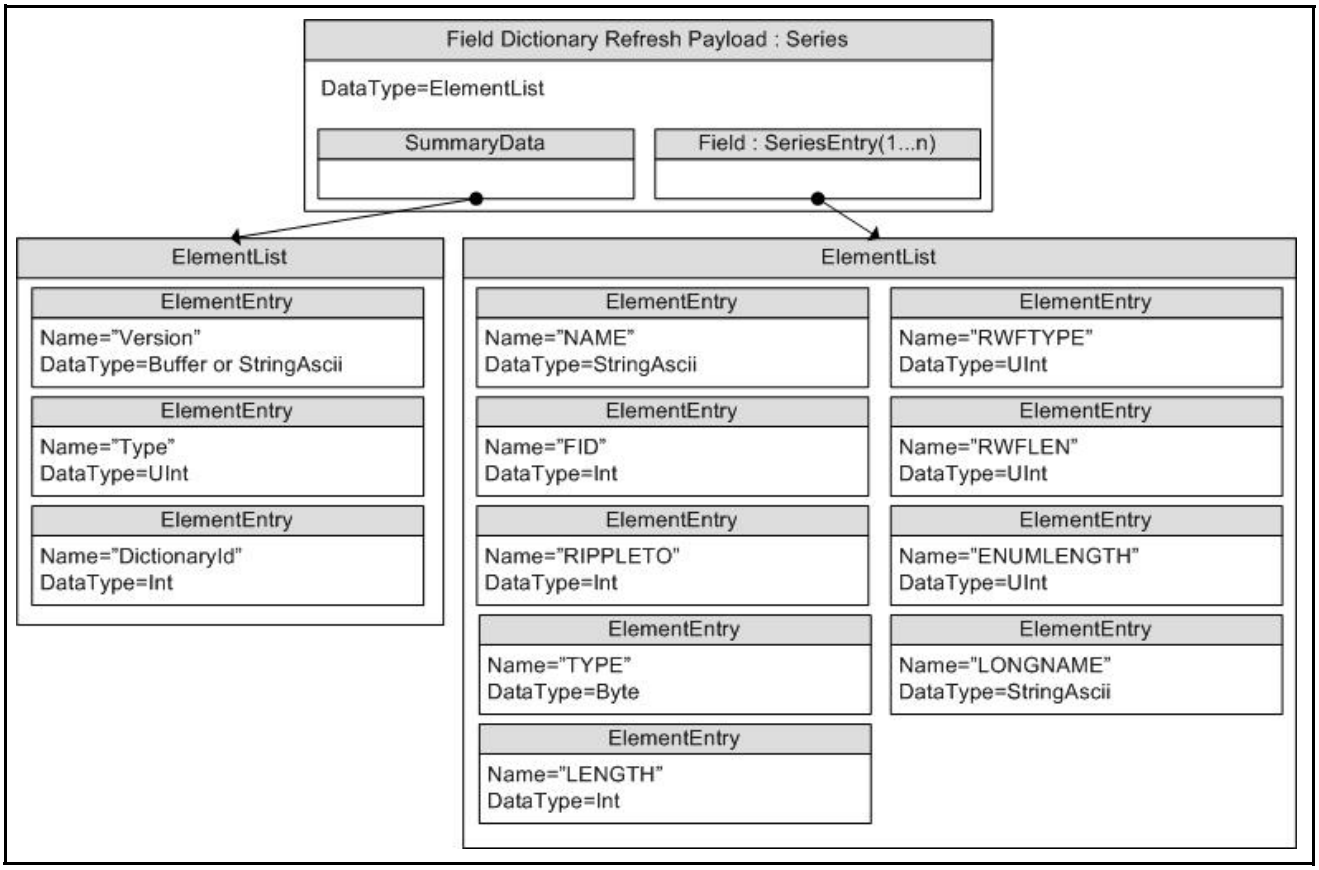


Figure 12. Field Dictionary Payload

Element entries do not have default values.

NAME	TYPE	LEAST VERBOSITY	RANGE/EXAMPLE	DESCRIPTION
NAME	ASCII	MINIMAL	e.g., “PROD_PERM”	Equivalent to the field’s **ACRONYM** (i.e., Short Name).
FID	Int	MINIMAL	-32768 to 32767	The field’s **`FieldId`**.
RIPPLETO	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 Message API C++ Edition Developers Guide*.
TYPE[a]	Int	MINIMAL	e.g., INTEGER	The data type of the field for the Marketfeed format.
LENGTH[a]	UInt	MINIMAL	0 to 65535	The maximum string length of the field for the Marketfeed format.
RWFTYPE	UInt	MINIMAL	e.g., Int	The data type (**`DataType`**) of the field.
RWFLEN	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	UInt	NORMAL	0 to 65535	Used for fields of type **Enum**. This is the length of the **DISPLAY** element in its Enumerated Types table (See Section 5.6.1).
LONGNAME	ASCII	NORMAL	e.g., “PERMISSION”	Equivalent to the field’s **DDE ACRONYM** (i.e., Long Name).

Table 38: 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
TIMACT      "TIME OF UPDATE"  5    NULL        TIME          5        TIME             5
TRDPRC_1    "LAST  "          6    TRDPRC_2    PRICE         17       REAL64           7
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 13. 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 14. 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_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 39: 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 (**`DataType`**) associated with the field.
RWF LEN	RWFLEN	A caching length hint associated with this field.

Table 40: Field Dictionary File Column Names and `ElementEntry` Names

5.5.2.3 RWFTYPE Keywords

The following keywords are supported for the RWFTYPE:

KEYWORD	DATA TYPE
ANSI_PAGE[a]	ANSI_Page
ARRAY[a]	Array
ASCII_STRING	ASCII
BUFFER	Buffer
DATE	Date
DATETIME[a]	DateTime
DOUBLE[a]	Double
ELEMENT_LIST, ELEM_LIST[a]	ElementList
ENUM	Enum
FIELD_LIST[a]	FieldList
FILTER_LIST[a]	FilterList
FLOAT[a]	Float
INT, INT32, INT64	Int
MAP[a]	Map
OPAQUE	Opaque
QOS[a]	QoS
REAL, REAL32, REAL64	Real
RMTES_STRING	RMTES
SERIES[a]	Series
STATUS[a]	Stream
TIME	Time
UINT, UINT32, UINT64	UInt
UTF8_STRING	UTF8
VECTOR[a]	Vector
XML[a]	XML

Table 41: Field Dictionary Type Keywords

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

5.5.2.4 FIELD TYPE Keywords

The **RDMFieldDictionary**'s **RWFTYPE** and **RWFLEN** are derived from the field dictionaries used in Marketfeed. Valid keywords for the Marketfeed Field Type are **INTEGER**, **ALPHANUMERIC**, **ENUMERATED**, **TIME**, **TIME_SECONDS**, **DATE**, or **PRICE**.

Thee following Refinitiv Wire Format types and values help ensure that data is not truncated when converted from Marketfeed to Refinitiv Wire Format. If converting Refinitiv Wire Format to Marketfeed, an Open Message Model provider application should ensure that Refinitiv Wire Format data does not overflow the Marketfeed length.

For **ALPHANUMERIC** types, if the data does not require RMTES, then the **ASCII_STRING** type should be used instead of the **RMTES_STRING** type.

Fields that cannot be converted to Marketfeed should have the Marketfeed type **NONE** and length **0**.

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 42: Marketfeed to Refinitiv Wire Format Mappings in RDMFieldDictionary

5.5.2.5 Custom FIDs

There are a couple of recommendations for custom Field IDentifiers:

FIELD TYPE	LENGTH	RWF TYPE	RWF LEN	NOTES
PRICE	17	REAL	9	**`Real`** can represent values with fractional denominators, trailing zeros, or up to 14 decimal positions.
INTEGER	15	INT	8	Signed integer value that has one sign bit and 63 value bits.

Table 43: Marketfeed to Refinitiv Wire Format Mappings in RDMFieldDictionary

5.5.3 Specific Usage: RDF Direct and FieldDefinition Dictionary

The FieldDefinition Dictionary provided by RDF Direct is named “RWFFld”. It has a **DictionaryId** of **1**.

All DataMasks are supported. DictionaryVerbose will return the same data as DictionaryNormal.

The response **`Payload.SummaryData`** includes Version, Type, and DictionaryId.

The RWFFld dictionary only uses the following types: INT32, INT64, INT, UINT32, UINT64, UINT, REAL32, REAL64, REAL, DATE, TIME, ENUM, BUFFER, ASCII_STRING, RMTES_STRING.

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 **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.

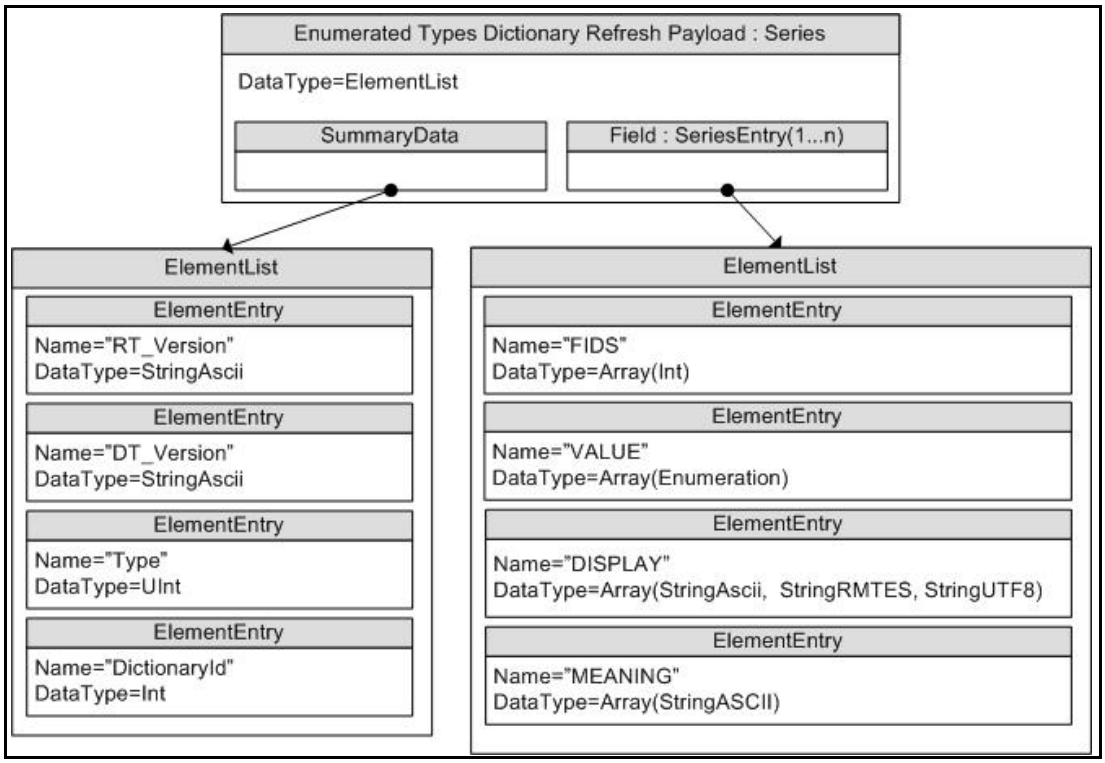


Figure 15. Enumerated Types Dictionary Refresh Message Payload

NAME	TYPE	LEAST VERBOSITY	EXAMPLE LIST	DESCRIPTION
FIDS	Array of Int	NORMAL	15, 1084, 1085,…	The **FieldId**'s of all fields that reference this table. These fields should have type **Enum** in the Field Dictionary and use the values given in the **VALUE** list. The **OmmArray.FixedWidth** should be 2 because each **FieldId** is a two-byte, signed integer value.
VALUE	Array of Enum	NORMAL	826, 840, …	Includes values that correspond to each Enumerated Type. **FieldEntries** that use the table contain these values. The **OmmArray.FixedWidth** should be 2 since each enum is a two-byte, unsigned integer value.
DISPLAY	Array of StringASCII, StringRMTES, or StringUTF8	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	Array of ASCII	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 44: 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 1: 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 2: 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_ENUM_TABLES = 2**. Other types are defined in Section 5.4. **Type** is used when encoding the domain representation of the field dictionary.

Table 45: 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 **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 46: 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	Quoted alphanumeric for the expanded string value. 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 47: Refinitiv Wire Format EnumType Dictionary File Values

5.6.3 Specific Usage: RDF Direct and EnumTable Dictionary

The RDF Direct EnumTable Dictionary uses the name “RWFEnum”. It has a **DictionaryId** of **1** to match the RWFFld Dictionary.

RDF Direct uses the standard file representation described in Section 5.5.2. The file does not include a **DictionaryId** or a Version number, so most existing enumtype.def parsers can parse the RWF FieldDictionary file without changes.

5.7 Special Semantics

5.7.1 DictionariesProvided and DictionariesUsed

The Directory's DirectoryInfo **FilterEntry** (refer to Section 4.3.1.1) includes two elements related to Dictionaries: **DictionariesProvided** and **DictionariesUsed**. Both elements contain an **Array** of **ASCII** dictionary names. These names can be used in **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 Directory.
2. Parse the **DictionariesProvided** from every service in the Directory.
3. Make a streaming request for any Dictionary from **DictionariesProvided** that is required to process or encode content.

NOTE: **DictionariesUsed** lists dictionaries that might be helpful or needed to encode, decode, cache, or display content from the dictionary provider; any additional dictionaries in this list might be acquired independently.

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 **Filter** requests a higher verbosity (e.g. **DICTIONARY_NORMAL**).
   - Version information can be used to determine if the consumer needs to update its dictionary.

5.7.2 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_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 **ASCII** 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

5.7.2.1 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.

5.7.2.2 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 **OmmState.Open**/ **OmmState.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.

5.8 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
DisplayTemplate	A ***DisplayTemplate*** dictionary contains specifications that describe how and where to display fields on a screen.
DataDefinition	A ***DataDefinition*** dictionary contains specifications for ElementListDefs and FieldListDefs that can be used for decoding FieldLists and ElementLists that have been optimized with SetDefinitions.
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 48: Other Dictionary Types

5.9 Specific Usage: RDMS

Refinitiv Real-Time Distribution System currently support only a single **`DictionaryId`**'s family. If the provider doesn't specify it then it is interpreted to be 1.

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 **MMT_MARKET_PRICE** Refinitiv Domain Model.

6.2 Usage

6.2.1 Market Price Request Message

A Market Price request message is encoded using **`ReqMsg`** 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 **ReqMsg.InterestAfterRefresh** to **true**. If the method is not set, the consumer requests a "snapshot," and the refresh ends the request (though updates might 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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_PRICE = 6**
Interactions	**Required**. • **`InitialImage`**: **true**, indicates that an initial image is required. • **`InterestAfterRefresh`**: **true**, indicates that a streaming request is required. • **`Pause`**: **true**, indicates that a pause is required.
Indications	Optional. **`ConflatedInUpdates`**: **true**, indicates conflated updates is required Batch and View request are specified in the **`Payload`**.
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).
Priority	Optional. Indicates the class and count associated with stream priority.
extendedHeader	Not used.
ServiceId	**Required**. Specifies the ID of the service from which the consumer wishes to request the item. **NOTE:** The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
NameType	Optional. When consuming from Refinitiv sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If unspecified, **`NameType`** defaults to **INSTRUMENT_NAME_RIC = 1**.

Table 49: Market Price Request Message

COMPONENT	DESCRIPTION / VALUE
Name	**Required**. Specifies the name of the requested item.
	NOTE: Not used for Batch Item request.
ServiceName	**Required**. Specifies the name of the service from which the consumer wishes to request the item.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. For more detailed information, refer to the Appendix A.

Table 49: Market Price Request Message (Continued)

6.2.2 Market Price Refresh Message

A Market Price Refresh Message is encoded using **RefreshMsg** and 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. The Market Price domain does not allow for multi-part refresh use. The provider should only send the **Name** and **ServiceName** in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_PRICE = 6**
State	**Required**. Includes the state of the stream and data.
Solicited	**Required**. Indicates whether the refresh was solicited. Possible settings are: • **true**: The refresh was solicited. • **false**: The refresh was unsolicited.
Indications	**Required.** Available settings include: • **Complete**: **true**, Indicates that the refresh is complete. • **DoNotCache**: **true**, Indicates that the refresh message should not be cached. • **ClearCache**: **true**, Indicates to clear the cache.
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.
ItemGroup	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies the permission information associated with content on this stream.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service from which the consumer wishes to request the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. **NameType** should match the **NameType** specified in the request. If unspecified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. This should match the requested name.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. This should consist of a **FieldList** containing all fields associated with the item.

Table 50: Market Price Refresh Message

6.2.3 Market Price Update Message

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

NOTE: The provider should only send the **Name** and **NameType** in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **NameType** must be provided for every Update response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_PRICE = 6**
UpdateTypeNum	**Required**. Indicates the general content of the update: • **INSTRUMENT_UPDATE_UNSPECIFIED = 0** • **INSTRUMENT_UPDATE_QUOTE = 1** • **INSTRUMENT_UPDATE_TRADE = 2** • **INSTRUMENT_UPDATE_NEWS_ALERT = 3** • **INSTRUMENT_UPDATE_VOLUME_ALERT = 4** • **INSTRUMENT_UPDATE_ORDER_INDICATION = 5** • **INSTRUMENT_UPDATE_CLOSING_RUN = 6** • **INSTRUMENT_UPDATE_CORRECTION = 7** • **INSTRUMENT_UPDATE_MARKET_DIGEST = 8** • **INSTRUMENT_UPDATE_QUOTES_TRADE = 9** • **INSTRUMENT_UPDATE_MULTIPLE = 10** • **INSTRUMENT_UPDATE_VERIFY = 11**
Indications	**Conditional.** • If **UpdateTypeNum** is set to be **INSTRUMENT_UPDATE_CORRECTION=7** or **UPDVERIFY**, **DoNotRipple** must be set to **true** • **DoNotCache**: **true**, Indicates the application should not cache this update message. • **DoNotConflate**: **true**, Indicates the application should not conflate updates.
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.
ConflatedCount	Optional. If a provider sends a conflated update, **ConflatedCount** specifies the number of updates in the conflation. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
ConflatedTime	Optional. If a provider sends a conflated update, **ConflatedTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
ItemGroup	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies permissioning information associated with only the contents of this update.
extendedHeader	Not used.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the data. **NOTE:** The application should set either the **ServiceName** or **ServiceId** of the service, but not both.

Table 51: Market Price Update Message

COMPONENT	DESCRIPTION / VALUE
NameType	**Conditional**. **`NameType`** is required if **MsgKeyInUpdates** was set to **true** on the request. **`NameType`** should match the name type specified on the request. If **`NameType`** is unspecified, its value defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **`Name`** is required if **MsgKeyInUpdates** was set to **true** on the request. **`Name`** specifies the name of the item being provided.
ServiceName	**Conditional**. **`ServiceName`** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. This should consist of a **`FieldList`** with any changed data.

Table 51: Market Price Update Message (Continued)

6.2.4 Market Price Status Message

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

NOTE: The provider should only send the **Name** and **NameType** in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **NameType** must be provided for every Status response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_PRICE = 6**
State	Optional. Specifies the current state information associated with the data and stream.
ItemGroup	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies permissioning information associated with only the contents of this message.
extendedHeader	Not used.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true** on the request. **NameType** should match the name type specified on the request. If **NameType** is unspecified, its value defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true** on the request. **Name** specifies the name of the item being provided.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 52: Market Price Status Message

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 Message API C++ Edition Developers Guide*.

6.3 Data: Response 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 by checking **`FieldEntry.LoadType`** and retrieving a specific type. For more information, refer to the *Enterprise Message API C++ Edition Developers Guide*.

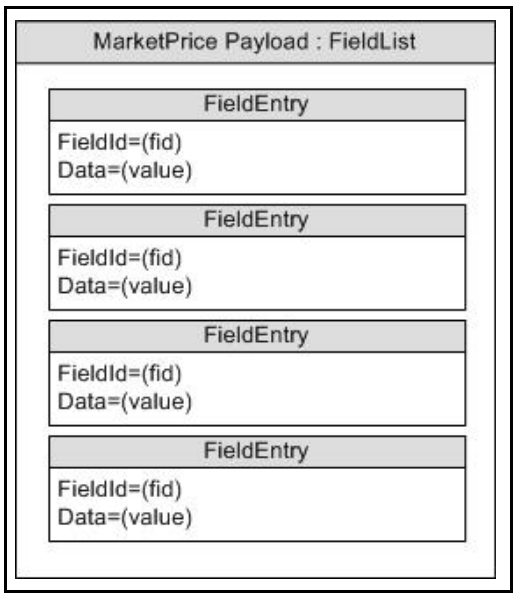


Figure 16. MarketPrice Response Message Payload

6.4 Special Semantics

6.4.1 Snapshots

MarketPrice is one of a few message model types that support a true snapshot. If a non-streaming request is made, then the **UpdateMsg** will not be sent. Status messages could be received before the single Refresh response message (for details refer to Section 2.3) is received. For streaming and snapshot streams, the Refresh response message will always be a single message and it will have **RefreshMsg.Complete** is set to **true**.

6.4.2 Ripple Fields

Some fields in a **FieldList** are defined as ripple fields. When the value of a ripple field changes, the former value automatically becomes the new value of another field. The change to the second field may, in turn, cause another field to be changed to reflect the second field's former value. Whether or not fields are rippled is determined by the value of **DoNotRipple**.

When a refresh message is received, all of the ripple fields delivered by the Venue/Exchange are present in the refresh message. However, the consuming application must set ripple behavior for fields not in the refresh message. In some cases, the values delivered for the "ripple-to" Fields in the refresh may be empty, but they must be present.

It is a responsibility of the Consumer application to ripple the Fields. The Enterprise Message API does NOT ripple fields on behalf of the consumer application. The Open Message Model **FieldList** concept supports rippling. However, the **FieldList** class does not cache, so it cannot ripple fields.

6.5 Specific Usage: RDF Direct MarketPrice

RDF Direct uses MarketPrice for SIAC Level 1, NASDAQ Level 1, and OPRA Level 1 data. The Refresh is provided in a single message. It contains all of the fields, even if they are blank.

6.6 Specific Usage: Legacy Records

MarketPrice can also be used for data structured like IDN records, such as:

- Page Records for reference page records and TS1 historical data.
- Chains for indices and ranked lists.
- Segment Chains for time & sales and news stories.

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 **MMT_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 using **`ReqMsg`** 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 **ReqMsg.InterestAfterRefresh** to **true**. If the method is not set, the consumer is requesting a "snapshot," and the refresh should end the request. 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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_ORDER = 7**
Interactions	**Required**. • **`InitialImage`**: **true**, Indicates that an initial image is required. • **`InterestAfterRefresh`**: **true**, Indicates that a streaming request is required. • **`Pause`**: **true**, Indicates that a pause is required.
Indications	Optional. **`ConflatedInUpdates`**: **true**, indicates conflated updates is required.
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).
Priority	Optional. Indicates the class and count associated with stream priority.
extendedHeader	Not used.
ServiceId	**Required**. This should be the ID associated with the service from which the consumer wants to request the item.
	NOTE: A consumer should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
ServiceName	**Required**. This should be the name of the service from which the consumer wishes to request data.
	NOTE: A consumer should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.

Table 53: Market By Order Request Message

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. When consuming from Refinitiv sources, **`NameType`** is typically set to **INSTRUMENT_NAME_RIC = 1** (the “Reuters Instrument Code”). If absent, the Enterprise Message API assumes a setting of **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. Specifies the requested item’s name.
	NOTE: Not used for Batch Item requests.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. For more detailed information, refer to Appendix A.

Table 53: Market By Order Request Message (Continued)

7.2.2 Market By Order Refresh Message

A Market By Order refresh message is encoded using **RefreshMsg** 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.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_ORDER = 7**
State	**Required**. The state of the stream and data.
Solicited	**Required**. Indicated whether the refresh was solicited. Available values are: • **true**: The refresh was solicited. • **false**: The refresh was unsolicited.
Indications	Optional. • **DoNotCache**: **true**, indicate do not cache this refresh message • **ClearCache**: **true**, indicate clear cache • **Complete**: **true**, indicate refresh complete
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.
ItemGroup	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies permission information associated with content on this stream.
extendedHeader	Not used.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The provider should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. **NameType** should match the **NameType** specified in the request. If absent, **NameType** is assumed to be **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. **Name** should match the requested item's name.
ServiceName	**Required**. Specifies the name of the service that provides the item.
	NOTE: The provider should set either the **ServiceName** or **ServiceId** of the service, but not both.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. An order book is represented by a **Map**, where each entry (**MapEntry**) contains information (**FieldList**) that corresponds to an order.

Table 54: Market By Order Refresh Message

7.2.3 Market By Order Update Message

A Market By Order update message is encoded using **UpdateMsg** 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. Updates may be received between the first Refresh and the RefreshComplete. It is the consuming application's responsibility to determine if the update is applicable to the data that has previously been sent in a refresh.

NOTE: The provider should only send the **Name** and **ServiceName** in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Update response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_ORDER = 7**
UpdateTypeNum	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED = 0** • **INSTRUMENT_UPDATE_QUOTE = 1**
Indications	Optional: • **DoNotCache**: **true**, Indicates that the application should not cache this update message. • **DoNotConflate**: **true**, Indicates that the application should not conflate this update message.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
ConflatedCount	Optional. If a provider sends a conflated update, **ConflatedCount** informs the consumer as to how many updates were included in the conflation. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
ConflatedTime	Optional. If a provider sends a conflated update, **ConflatedTime** informs the consumer as to the interval (in milliseconds) over which data was conflated. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
PermissionData	Optional. **PermissionData** contains permissioning information associated only with the contents of this update.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**. **ServiceName** specifies the name of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**. **ServiceId** specifies the ID of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. **NameType** must match the name type in the item's request message (typically **INSTRUMENT_NAME_RIC = 1**).
Name	Optional (Required if **MsgKeyInUpdates** was set to **true**). **Name** specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.

Table 55: Market By Order Update Message

COMPONENT	DESCRIPTION / VALUE
Attrib	Not used.
Payload	**Required**. The order book is represented by a **`Map`**, where each map entry (**`MapEntry`**) holds information (**`FieldList`**) corresponding to an order.

Table 55: Market By Order Update Message (Continued)

7.2.4 Market By Order Status Message

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

NOTE: The provider should only send the **Name** and **ServiceName** in the first Refresh response message. However **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Status response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_ORDER = 7**
State	Optional. Specifies the current state information associated with the data and stream.
Indications	Optional: **ClearCache**: **true**, Indicates to clear the cache.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
ItemGroup	Optional. The provider may use this to change the item's **ItemGroup** (for details, refer to Section 4.3.1.3).
PermissionData	Optional. **PermissionData** specifies any new permissioning information associated with all of the stream's contents.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**). Specifies the name of the service providing data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**). **ServiceId** specifies the ID of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. **NameType** must match the name type in the item's request message. If not specified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (Required if **MsgKeyInUpdates** was set to **true**). **Name** specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 56: Market By Order Status Message

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 Message API C++ Edition Developers Guide*.

7.3 Data

7.3.1 Response Message Payload

The payload is a **`Map`**. Refreshes for this **`Map`** may be in multiple response messages. The bandwidth of the refresh messages can be optimized by putting multiple **`MapEntry`** in each response messages. For optimal performance the packed map entries in each response message should use less than 6000 bytes. If the data is split into multiple response messages, then a **`Map.TotalCountHint`** should be provided to optimize downstream caching. Because the fields in each **`MapEntry`** are the same, bandwidth can be further optimized by DataDefinitions.

NOTE: There are two possible usage scenarios:

- Pattern 1: ORDER_PRC, ORDER_SIDE and ORDER_SIZE.
- Pattern 2.1: BID and BIDSIZE, or Pattern 2.2: ASK and ASKSIZE.

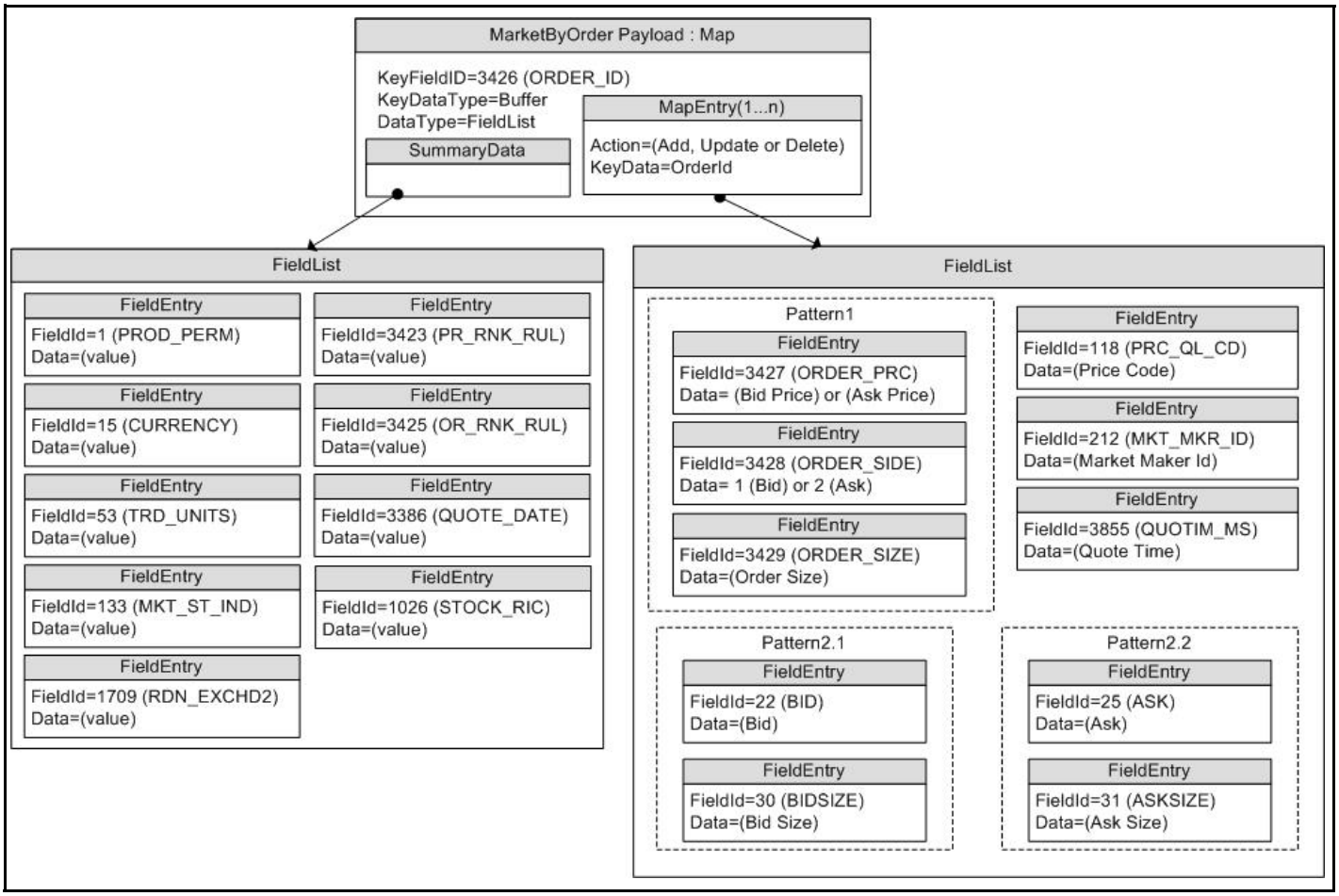


Figure 17. MarketByOrder Response Message Payload

7.3.2 Summary Data

The **Map.SummaryData** 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 (**QUOTE_DATE**)
- RIC of the underlying equity (**STOCK_RIC**)

7.3.3 MapEntry Contents

The **MapEntry.Key** is a Buffer, Ascii, or Rmtes that contains the Order ID. The **MapEntry.KeyFieldId** may be set to ORDER_ID, so the information does not have to be repeated in the **MapEntry.Value**.

The **MapEntry.Value** is a **FieldList** that typically contains the following information about the order:

- 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 Special Semantics

None.

7.5 Specific Usage: RDF Direct and Response Message Payload

RDF Direct uses MarketByOrder for several markets, including NASDAQ TotalView, Archipelago ECN order book, and Instinet ECN order book.

The payload is a **Map**. Each Refresh for this **Map** includes summary data and a single **MapEntry**. Updates are not sent for any map entry until after the message is sent with **RefreshMsg.Complete** set to **true**. Since each resposne message includes only one map entry, DataDefinitions are not used to reduce bandwidth. The **Map.TotalCountHint** is not provided.

The **Map.SummaryData** is sent in every Refresh, even if it does not change. The fields used are from the RWFFld Field Dictionary:

- PROD_PERM (1): Integer for permission information.
- CURRENCY (15): Enumeration of currency for the orders.
- TRD_UNITS (53): Enumeration of trade units for the precision for which order prices are set.
- MKT_ST_IND (133): Enumeration of market state.
- RDN_EXCHD2 (1709): Enumeration of exchange on which the orders were placed.
- PR_RNK_RUL (3423): Enumeration of price ranking rules.
- OR_RNK_RUL (3425): Enumeration of order ranking rules.

- STOCK_RIC (1026): RIC of the underlying equity.

The **MapEntry.Key** is a buffer that contains the Order ID. The **Map.KeyFieldId** is not set, but this may be changed in the future.

The **MapEntry.Data** is a field list that contains some or all of the following information about the order:

- ORDER_PRC (3427) & ORDER_SIDE (3428): Real and Enumeration for the order price & side (buy or sell/bid or ask).
- ORDER_SIZE (3429): Real for the order size.
- ORDER_ID (3426): Same value as the **MapEntry.KeyData**. This may be removed in the future by setting the **Map.KeyFieldId** to ORDER_ID (3426).
- QUOTIM_MS (3855): Quote Time in millisecond since GMT of the current day in the GMT time zone.

The **FieldList.DictId** is **0**, so it should be ignored.

7.6 Specific Usage: RDMS

For the most part, MarketByOrder data from Refinitiv Real-Time Distribution System is the same as it is from the original source of the data (e.g., Refinitiv Data Feed Direct). However, if caching is enabled in an Refinitiv Real-Time Distribution System component, there are two differences.

- The number of messages packed into each Refresh response message may be different.
- An updated response message might be delivered between Refresh response messages and before the message with **RefreshMsg.Complete** set **true**. It is the consumer applications responsibility to apply the indicated changes.

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 **MMT_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 using **`ReqMsg`** 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 **ReqMsg.InterestAfterRefresh** to **true**. 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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
Interactions	**Conditional**. • **`InitialImage`**: **true**, indicates that an initial image is required. • **`InterestAfterRefresh`**: **true**, indicates that a streaming request is required. • **`Pause`**: **true**, indicates that a pause is required.
Indications	Optional. **`ConflatedInUpdates`**: **true**, indicates that conflated updates are required. Batch and View requests are specified in the Payload.
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).
Priority	Optional. Indicates the class and count associated with stream priority.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service from which the consumer wishes to request data.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the requested item.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.

Table 57: Market By Price Request Message

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. Typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code") when consuming from Refinitiv sources. If absent, its default value is **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. Specifies the name of the requested item.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. For further details, refer to the Appendix A.

Table 57: Market By Price Request Message (Continued)

8.2.2 Market By Price Refresh Message

A Market By Price refresh message is encoded using **`RefreshMsg`** 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.

NOTE: The provider should send **`Name`** and **`ServiceName`** only in the first Refresh response message, unless **MsgKeyInUpdates** is set to **true**, in which case **`Name`** and **`ServiceName`** must be provided in each Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
State	**Required**. Indicates the state of the stream and data.
Solicited	**Required**. Indicates whether the refresh message was solicited. • **true**: The refresh message is solicited. • **false**: The refresh message is unsolicited.
Indications	**Conditional**. • **`DoNotCache`**: **true**, indicates that the application should not cache this refresh message. • **`ClearCache`**: **true**, indicates that the application should clear its cache. • **`Complete`**: **true**, indicates that this is the last message in the refresh complete.
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.
ItemGroup	Optional. Associates the item with an Item Group (for further information, refer to Section 4.3.1.3).
PermissionData	Optional. If present, specifies permission information associated with the stream's content.
extendedHeader	Not used.

Table 58: Market By Price Refresh Message

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. Specifies the name of the service that provides the item.
	NOTE: The consumer application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The consumer application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. **NameType** should match the **NameType** specified in the request. If absent, this value is assumed to be **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. **Name** should match the name specified in the request.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. The order book is represented by a **Map**, where each entry (**MapEntry**) contains a **FieldList** which has information about a price point.

Table 58: Market By Price Refresh Message (Continued)

8.2.3 Market By Price Update Message

A Market By Price update message is encoded using **UpdateMsg** 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. Updates will not be received before images. True snapshots are supported.

NOTE: The provider should send **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then **Name** and **ServiceName** must be provided for every Update response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
UpdateTypeNum	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED = 0** • **INSTRUMENT_UPDATE_QUOTE = 1**
Indications	Optional. • **DoNotCache**: **true**, indicates to not cache the update message. • **DoNotConflate**: **true**, indicates to not conflate the update message.
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.
ConflatedCount	Optional. If a provider sends a conflated update, **ConflatedCount** specifies how many updates were included in the conflation. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** flag in the request.
ConflatedTime	Optional. If a provider sends a conflated update, **ConflatedTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** flag in the request.
PermissionData	Optional. Specifies permissioning information for the update's content.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the item.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true** on the request. **NameType** should match the **NameType** specified in the item's request message. If **NameType** is not specified, it uses the default **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true** on the request) Specifies the name of the item being provided.
Filter	Not used.

Table 59: Market By Price Update Message

COMPONENT	DESCRIPTION / VALUE
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. MarketByPrice is represented by a **Map**, where each entry contains a **FieldList** containing information about a price point.

Table 59: Market By Price Update Message (Continued)

8.2.4 Market By Price Status Message

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

NOTE: The provider should send **Name** and **ServiceName** only in the first Refresh response message, unless **MsgKeyInUpdates** is set to **true**, in which case **Name** and **ServiceName** must be provided for every Status response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
State	Optional. Specifies current state information associated with the data and stream.
Indications	Optional. **ClearCache**: **true**, Indicates to clear the cache.
QoS	Optional. Specifies the QoS at which the stream is provided.
ItemGroup	Optional. Specifies the item's **ItemGroup** (the provider can use this component to change the item's **ItemGroup**).
PermissionData	Optional. Specifies new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the item. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true** on the request. **NameType** should match the **NameType** specified in the item's request message. If **NameType** is not specified, it uses the default **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.

Table 60: Market By Price Status Message

COMPONENT	DESCRIPTION / VALUE
Attrib	Not used.
Payload	Not used.

Table 60: Market By Price Status Message (Continued)

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 Message API C++ Edition Developers Guide*.

8.3 Data

8.3.1 Response Message Payload

The payload is a **Map**. Refreshes for this **Map** may be in multiple Response messages. The bandwidth of the refresh messages can be optimized by putting multiple **MapEntry** in each response message. For optimal performance the packed map entries in each response message should use less than 6000 bytes. If the data is split into multiple messages, then a **Map.TotalCountHint** should be provided to optimize downstream caching. Since the fields in each map entry are the same, bandwidth can be further optimized by DataDefinitions.

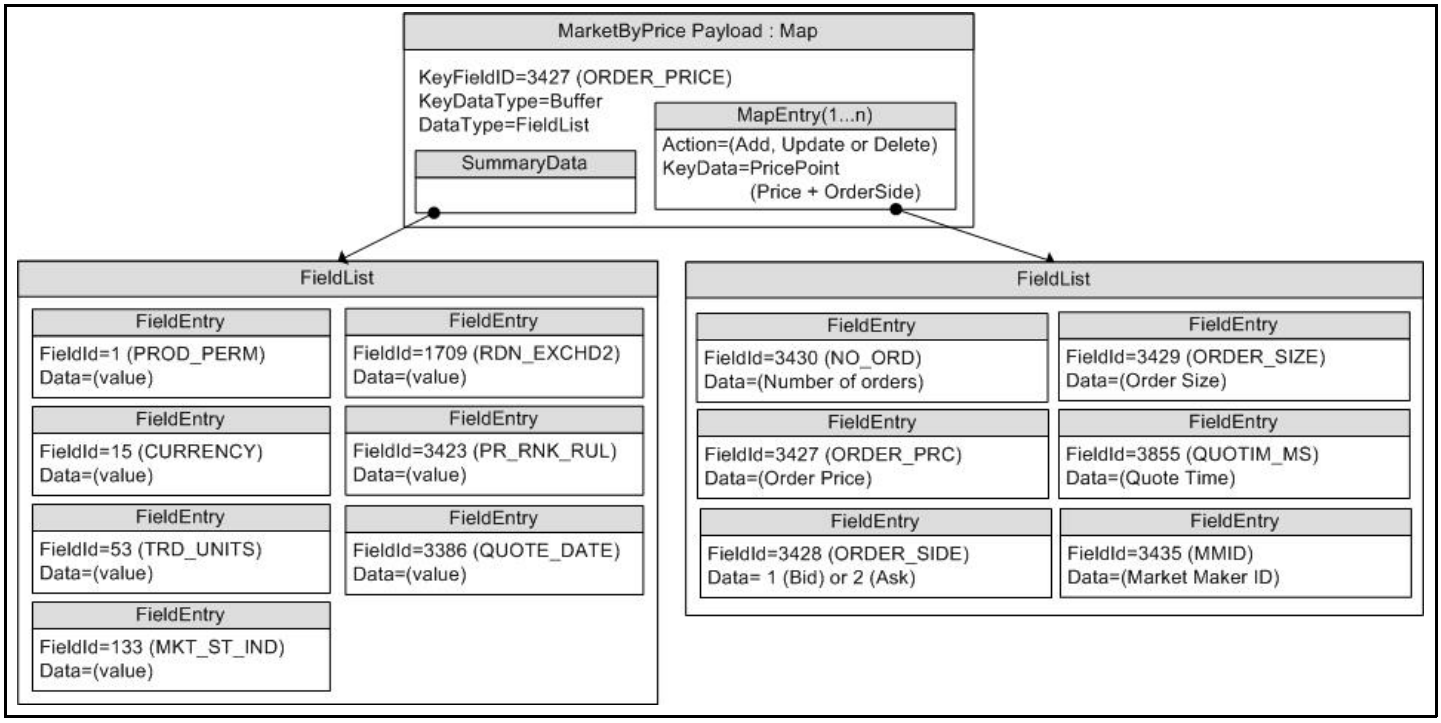


Figure 18. MarketByPrice Response Message Payload

8.3.2 Summary Data

The **Map.SummaryData** 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.Key Contents

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

MapEntry.Data is a **FieldList** that contains some or all of the following information about the price point:

- 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 Special Semantics

None

8.5 Specific Usage: RDF Direct and the Response Message Payload

RDF Direct uses MARKET_BY_PRICE for several markets, including NYSE OpenBook, Archipelago ECN market depth, and Instinet ECN market depth.

The payload is a **Map**. Each refresh message for this **Map** includes SummaryData and up to 50 map entries. Updates are not sent for any map entry until after the **RefreshMsg.Complete** is set to **true**. DataDefinitions are not used to reduce bandwidth. **Map.TotalCountHint** is not provided.

Map.SummaryData is sent in every refresh message, even if it does not change. The fields used are from the RWFFld Field Dictionary:

- PROD_PERM (1): Integer for permission information
- CURRENCY (15): Enumeration of currency for the orders
- TRD_UNITS (53): Enumeration of trade Units for the precision for which order prices are set
- MKT_ST_IND (133): Enumeration of market state
- RDN_EXCHD2 (1709): Enumeration of exchange on which the orders were placed

The **MapEntry.Key**'s data is a Buffer that contains the combination of price and order side (**B** for buy or **S** for Sell), so each key is unique within its map. The **MapEntry.Key**''s data should be treated as a single entity and is not meant to be parsed.

The **MapEntry.Value** is a **FieldList** that contains the following information about the price point:

- NO_ORD (3430): Integer for the Number of Orders aggregated into this **MapEntry**
- ORDER_PRC (3427) & ORDER_SIDE (3428): Real and Enumeration for the order price & side (buy or sell/bid or ask)
- ORDER_SIZE (3429): Real for the aggregated size of the order at this price
- QUOTIM_MS (3855): Quote Time in millisecond since GMT of the current day in the GMT time zone
- Some venues may provide an extra field that contains a map. The **MapEntry.KeyData** will have a **KeyFieldId** which is MMID (3435). If the positions of each market maker are available, then the MapEntry.Value will contain a **FieldList**. The field list will contain a single field with the position of that market maker. If positions for each market maker are not available, **MapEntry.Value**'s data type will be NoData.

The **FieldList.DictId** is **0**, so it should be ignored.

8.6 Specific Usage: RDMS

For the most part, MarketByPrice data from Refinitiv Real-Time Distribution System is the same as it is from the original source of the data (e.g. RDF Direct). However, if caching is enabled in an Refinitiv Real-Time Distribution System component, there are two differences.

- The number of messages packed into each Refresh response message may be different.
- Updated response messages may be delivered between Refresh response messages, before the **RespMsg.Complete** is set to **true**. It is the consumer applications responsibility to apply the indicated changes.

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 **MMT_MARKET_MAKER** Refinitiv Domain Model.

9.2 Usage

9.2.1 Market Maker Request Message

A Market Maker request message is encoded using **`ReqMsg`** 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 **ReqMsg.InterestAfterRefresh** to **true**. If the flag is not set, the consumer requests a "snapshot," and the final part of the refresh 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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_MAKER = 9**
Interactions	**Conditional**. Use the appropriate interactions according to your messaging needs: • **`InitialImage`**: **true**, indicates that an initial image is required. • **`InterestAfterRefresh`**: **true**, indicates that a streaming request is required. • **`Pause`**: **true**, indicates that a pause is required.
Indications	Optional. **`ConflatedInUpdates`**: **true**, indicates that conflated updates are required. Batch and View request are specified in the **`Payload`**.
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).
Priority	Optional. Indicates the class and count associated with stream priority.
extendedHeader	Not used.
ServiceName	**Required.** Specifies the name of the service from which the consumer wishes to request the item.
	NOTE: The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.

Table 61: Market Maker Request Message

COMPONENT	DESCRIPTION / VALUE
ServiceId	**Required**. Specifies the ID of the service that provides the requested item.
	NOTE: The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
NameType	Optional. When consuming from Refinitiv sources, **`NameType`** is typically set to **INSTRUMENT_NAME_RIC = 1** (the “Reuters Instrument Code”). If absent, its value reverts to the default, which is **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. Specifies the name of the requested item.
	NOTE: Not used for Batch Item request.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. For more details, refer to Appendix A.

Table 61: Market Maker Request Message (Continued)

9.2.2 Market Maker Refresh Message

A Market Maker refresh message is encoded using **RefreshMsg** 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.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_MAKER = 9**
State	**Required**. Indicates the state of the stream and data.
Solicited	**Required**. Indicates whether the refresh was solicited. Available values are: • **true**: The message was solicited. • **false**: The message was unsolicited.
Indications	**Conditional**. • **DoNotCache**: **true**, indicates that the application should not cache. • **ClearCache**: **true**, indicates that the application should clear the cache. • **Complete**: **true**, indicates that the message is the final one in the refresh.
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.
ItemGroup	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies permission information associated with this stream's content.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service that provides the item.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. **NameType** should match the **NameType** specified in the request. If absent, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. A symbol for the Market Maker item.
Filter	Not used.
Identifier	Not used.
Payload	**Required**. A Market Maker is represented by a **Map**, where each entry (**MapEntry**) contains an **FieldList** which has information about a market maker.

Table 62: Market Maker Refresh Message

9.2.3 Market Maker Update Message

A Market Maker update message is encoded using **UpdateMsg** and sent by Open Message Model interactive provider and non-interactive provider applications. Updates will not be received before images, and a true snapshot is supported.

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

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Update response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_MAKER = 9**
UpdateTypeNum	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED = 0** • **INSTRUMENT_UPDATE_QUOTE = 1**
Indications	Optional: • **DoNotCache**: **true**, specifies that the update message should not be cached. • **DoNotConflate**: **true**, specifies that the update message should not be conflated.
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.
ConflatedCount	Optional. If a provider sends a conflated update, **ConflatedCount** specifies how many updates are in the conflation. The consumer indicates interest in this information by setting **ReqMsg.ConflatedInUpdates** to true in the request.
ConflatedTime	Optional. If a provider sends a conflated update, **ConflatedTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting **ReqMsg.ConflatedInUpdates** to true in the request.
PermissionData	Optional. Specifies permissioning information associated only with the contents of this update.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**. **ServiceName** specifies the name of the service that provides the data.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**. **ServiceId** specifies the ID of the service that provides the item.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. **NameType** must match the name type in the item's request message (typically **INSTRUMENT_NAME_RIC = 1**). If absent, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.

Table 63: Market Maker Update Message

COMPONENT	DESCRIPTION / VALUE
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true**. **Name** specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. A Market Maker is represented by a **Map**, where each entry (**MapEntry**) contains a **FieldList** which in turn contains information about a market maker.

Table 63: Market Maker Update Message (Continued)

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.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Status response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_MAKER = 9**
State	Optional. Specifies current state information associated with the data and stream.
Indications	Optional. **ClearCache**: **true**, indicates that the application should clear the cache.
QoS	Optional. Specifies the QoS at which the stream is provided.
ItemGroup	Optional. The provider can use this component to change the items' **ItemGroup**.
PermissionData	Optional. Specifies new permissioning information associated with all of the stream's contents.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**. **ServiceName** specifies the name of the service that provides the data.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**. **ServiceId** specifies the ID of the service that provides the item.
	NOTE: The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. **NameType** must match the name type in the item's request message (typically **INSTRUMENT_NAME_RIC = 1**). If absent, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true**. **Name** specifies the name of the item being provided.

Table 64: Market Maker Status Message

COMPONENT	DESCRIPTION / VALUE
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 64: Market Maker Status Message (Continued)

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 Message API C++ Edition Developers Guide*.

9.3 Data

9.3.1 Response Message Payload

The payload is a **Map**. Refreshes for this **Map** may be in multiple response messages. The bandwidth of the Refresh response messages can be optimized by putting multiple **MapEntry** in each Response message. For optimal performance the packed map entries in each response message should use less than 6000 bytes. If the data is split into multiple messages, then a **Map.TotalCountHint** should be provided to optimize downstream caching. Because the fields in each map entry are identical, bandwidth can be further optimized by DataDefinitions.

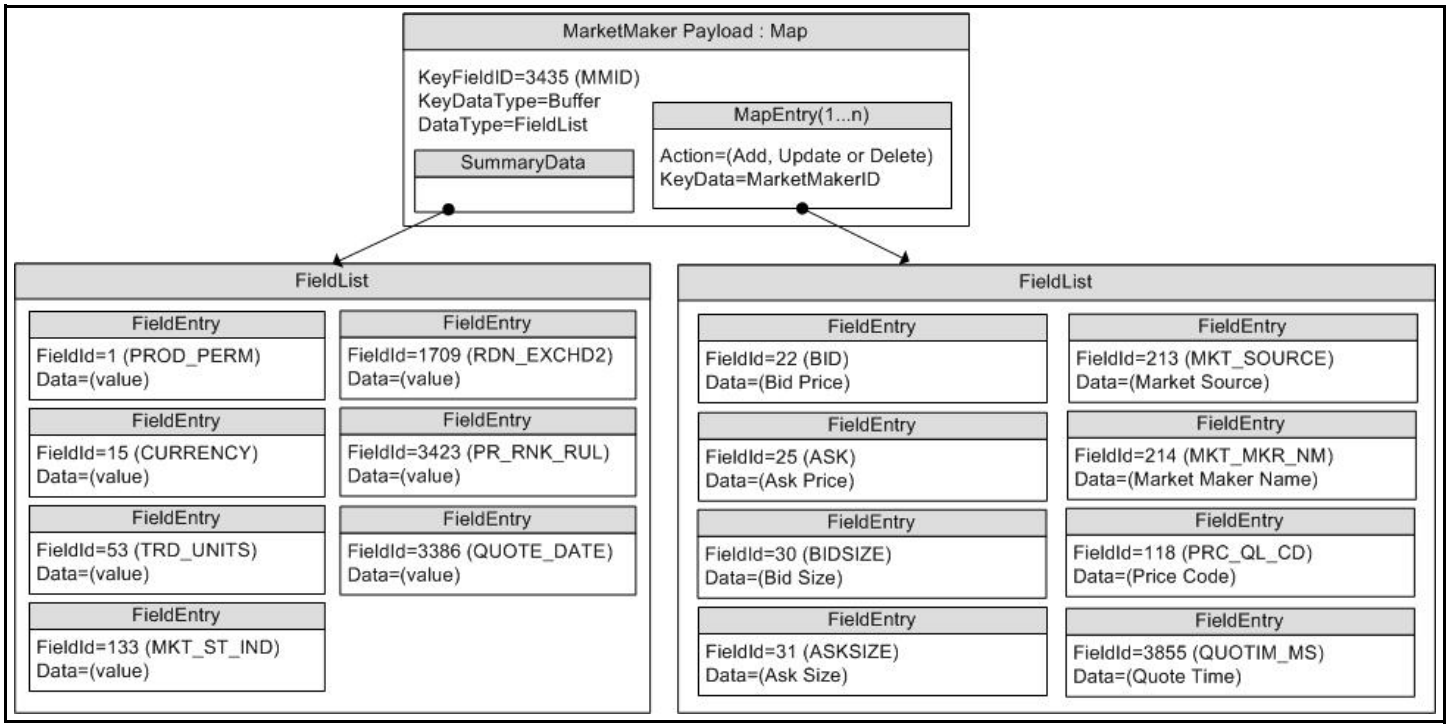


Figure 19. MarketMaker Response Message Payload

9.3.2 Summary Data

The **Map.SummaryData** only needs to be present in the first refresh part. Typical fields in the **Map.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**'s data is a **Buffer** that contains a unique market maker's ID. The **Map.KeyFieldId** may be set to MMID or MKT_MKR_ID, so the information does not have to be repeated in the **MapEntry.Data**.

Each **MapEntry** houses a **FieldList** that contains information about the market maker.

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 Special Semantics

None.

9.5 Specific Usage: RDF Direct and the Response Message Payload

RDF Direct uses MARKET_MAKER for NASDAQ Market Makers.

The payload is a **Map**. Each Refresh message for this **Map** includes SummaryData and up to 50 **MapEntry**s. Updates are not sent for any map entry until after the **RefreshMsg.Complete** is sent with a value of **true**. DataDefinitions are not used to reduce bandwidth. The **Map.TotalCountHint** is not provided.

Map.SummaryData is sent in every refresh, even if it does not change. The fields used are from the RWFFld Field Dictionary:

- PROD_PERM (1): Integer for permission information
- CURRENCY (15): Enumeration of currency for the orders
- TRD_UNITS (53): Enumeration of trade Units for the precision for which order prices are set
- MKT_ST_IND (133): Enumeration of market state
- RDN_EXCHD2 (1709): Enumeration of exchange on which the orders were placed
- PR_RNK_RUL (3423): Enumeration of price ranking rules

The **MapEntry.Key**'s Data is a Buffer containing a unique market maker ID. The **MapEntry.KeyFieldId** is not set, but this may be changed in the future.

The **MapEntry.Data** is a **FieldList** that contains some or all of the following information about the order:

- BID (22): Real with the best bid price from this market maker
- ASK (25): Real with the best ask price from this market maker
- BIDSIZE (30): Real with the size of the best bid
- ASKSIZE (31): Real with the size of the best ask
- MKT_MKR_ID (212): RmtesString with the Market Maker ID. This may be removed in the future by setting the **Map.KeyFieldId** to MKT_MKR_ID (212) or MMID (3435).
- MKT_SOURCE (213): Enumeration with the Exchange or City of the quote
- MKT_MKR_NM (214): RmtesString with the Market Maker Name
- PRC_QL_CD (118): Enumeration for first price qualifier
- PRC_QL2 (131): Enumeration for second price qualifier
- QUOTIM_MS (3855): Quote Time in millisecond since GMT of the current day in the GMT time zone

The **FieldList.DictId** is **0**, so it should be ignored.

9.6 Specific Usage: RDMS

For the most part, MarketMaker data from Refinitiv Real-Time Distribution System is the same as it is from the original source of the data (e.g., Refinitiv Data Feed Direct). However, if caching is enabled in an Refinitiv Real-Time Distribution System component, there will be two differences.

The number of messages packed into each Refresh response message may be different.

An Update response message may be delivered between Refresh response messages, before **RefreshMsg.Complete** is sent with a **true** value. It is up to the consumer application to apply the indicated changes.

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 **MMT_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 using **`ReqMsg`** 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 **ReqMsg.InterestAfterRefresh** to **true**. If the flag is not set, the consumer requests a "snapshot," and the final part of the refresh (i.e., the refresh has the **RefreshMsg.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 Message API C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_YIELD_CURVE = 22**
Interactions	**Conditional**. • **`InitialImage`**: **true**, requests an initial image. • **`InterestAfterRefresh`**: **true**, requests streaming updates. • **`Pause`**: **true**, requests that the application pause the item.
Indications	Optional. **`ConflatedInUpdates`**: **true**, requests that the application send conflated updates. Batch and View request are specified in the **`Payload`**.
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).
Priority	Optional. Indicates class and count associated with stream priority.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service from which the consumer wishes to request the item.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.

Table 65: Yield Curve Request Message

COMPONENT	DESCRIPTION / VALUE
ServiceId	**Required**. Specifies the ID of the service that provides the requested item.
	NOTE: The application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
NameType	Optional. When consuming from Refinitiv sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If this is not specified, **`NameType`** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**.Specifies the name of the requested item.
	NOTE: Not used for Batch Item request.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When leveraging such features as View or Batch, the payload can contain information relevant to that feature. For more information, refer to Appendix A.

Table 65: Yield Curve Request Message (Continued)

10.2.2 Yield Curve Refresh Message

A Yield Curve Refresh Message is encoded using **RefreshMsg** and 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.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_YIELD_CURVE = 22**
State	**Required**. Includes the state of the stream and data.
Solicited	**Required**. Indicates whether the refresh was solicited. Available values are: • **true**: The message was solicited. • **false**: The message was unsolicited.
Indications	**Conditional**. • **DoNotCache**: **true**, indicates that the application should not cache this refresh message. • **ClearCache**: **true**, indicates that the application should clear the cache. • **Complete**: **true**, indicates that the message is the final one in the refresh.
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.
ItemGroup	**Required**. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies permission information associated with content on this stream.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. Should match the **NameType** specified in the request. If this is not specified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. This should match the requested name.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.

Table 66: Yield Curve Refresh Message

COMPONENT	DESCRIPTION / VALUE
Payload	**Required**. This should consist of a **FieldList** containing all fields associated with the item. Some **FieldEntry**s 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.

Table 66: Yield Curve Refresh Message (Continued)

10.2.3 Yield Curve Update Message

A Yield Curve Update Message is encoded using **UpdateMsg** and sent by Open Message Model provider and non-interactive provider applications. It conveys any changes to an item's data. Updates may be received between the first Refresh and the RefreshComplete. It is the consuming application's responsibility to determine if the update is applicable to the data that has previously been sent in a refresh.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_YIELD_CURVE = 22**
UpdateTypeNum	**Required**. Indicates the general content of the update. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED = 0** • **INSTRUMENT_UPDATE_QUOTE = 1**
Indications	**Conditional**. • **DoNotCache**: **true**, indicates that the application should not cache this update message. • **DoNotConflate**: **true**, indicates that the application should not conflate the update message.
SeqNum	Optional. A user-specified, item-level sequence number which the application can use to sequence messages in this stream.
PartNum	Not used.
ConflatedCount	Optional. If the provider sends a conflated update, **ConflatedCount** specifies how many updates are in the conflation. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
ConflatedTime	Optional. If a provider is sending a conflated update, **ConflatedTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** to **true** in the request.
PermissionData	Optional. Permissioning information associated with only the contents of this update.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.

Table 67: Yield Curve Update Message

COMPONENT	DESCRIPTION / VALUE
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true** on the request. Should match the **NameType** specified on the request. If this is not specified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	**Required**. This should consist of a **FieldList** containing all fields associated with the item. Some **FieldEntry**s 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.

Table 67: Yield Curve Update Message (Continued)

10.2.4 Yield Curve Status Message

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

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Status response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_YIELD_CURVE = 22**
State	Optional. Current state information associated with the data and stream.
Indications	Optional. **ClearCache**: **true**, indicates that the cache should be cleared.
QoS	Optional. Specifies the QoS at which the stream is provided.
ItemGroup	Optional. The provider can use this component to change the item's **ItemGroup**.
PermissionData	Optional. Specifies new permissioning information associated with all contents on the stream.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the name of the service that provides the data.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true** on the request. Specifies the ID of the service that provides the item.
	NOTE: The application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true** on the request). Should match the **NameType** specified on the request. If this is not specified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.

Table 68: Yield Curve Status Message

COMPONENT	DESCRIPTION / VALUE
Name	**Conditional**. **`Name`** is required if **MsgKeyInUpdates** was set to **true** on the request). Specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 68: Yield Curve Status Message (Continued)

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 Message API C++ Edition Developers Guide*.

10.3 Data

10.3.1 Response Message 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. Because the fields in each **`Vector`** are the same, bandwidth can be further optimized by DataDefinitions.

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.

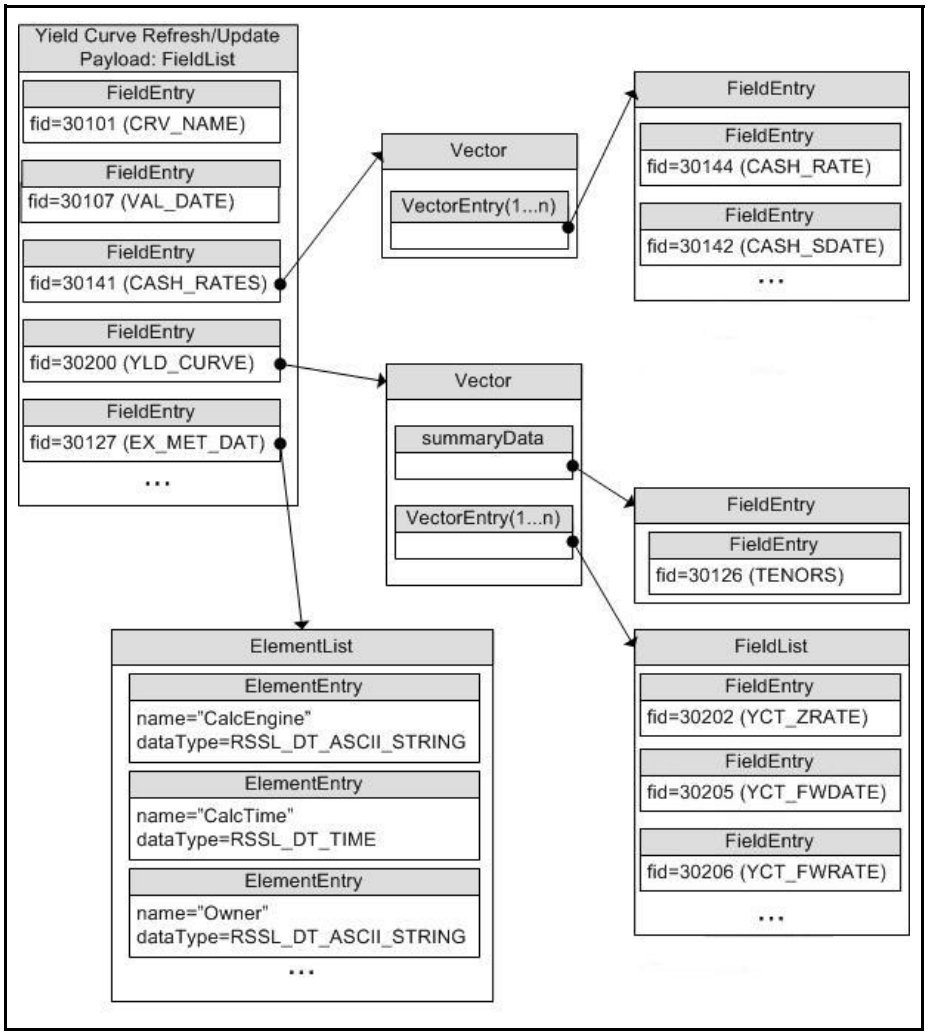


Figure 20. Yield Curve Payload Example

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 by checking the **FieldEntry** data type.

- 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 Message API C++ Edition Developers Guide*.

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

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 basis (**CASH_BASIS**).
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 basis (**FUTR_BASIS**).
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 69: Yield Curve Inputs and Outputs

10.4 Special Semantics

None

10.5 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.

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`** that contains additional cross-reference information such as permission information, name type, or other venue-specific content.

NOTE: **`GenericMsg`**(s) are not supported for **MMT_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 **ReqMsg.InterestAfterRefresh** to **true**) to receive updates, typically associated with item additions or removals from the list.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_SYMBOL_LIST = 10**
Interactions	**Conditional**. • **`InitialImage`**: **true**, indicates that an initial image is required. • **`InterestAfterRefresh`**: **true**, indicates that a streaming request is required. • **`Pause`**: **true**, indicates that a pause is required.
Indications	Optional. **`ConflateInUpdates`**: **true**, indicates that conflated updates are required. Batch and View requests are specified in the **`Payload`**.
QoS	Not used.
worstQos	Not used.
Priority	Optional. Indicates class and count associated with stream priority.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service from which the consumer wants to request the item.
	NOTE: The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the requested item.
	NOTE: The consumer application should set either the **`ServiceName`** or **`ServiceId`** of the service, but not both.
NameType	Optional. **`NameType`** should match name type specified in the request. When consuming from Refinitiv sources, **`NameType`** is typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If absent, **`NameType`** defaults to **INSTRUMENT_NAME_RIC = 1**.

Table 70: Symbol List Request Message

COMPONENT	DESCRIPTION / VALUE
Name	**Required**. Specifies the name of the requested item.
	NOTE: Not used for Batch Item requests.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Optional. When leveraging such features as View, Batch, or behaviors related to the Symbol List Request, the payload can contain information relevant to that feature. For more detailed information, refer to Appendix A.

Table 70: Symbol List Request Message (Continued)

11.2.2 Symbol List Refresh Message

A Symbol List refresh Message is encoded using **RefreshMsg** and 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.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Refresh response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_SYMBOL_LIST = 10**
State	**Required**. Indicates the state of the stream and data.
Solicited	**Required**. Indicates whether the refresh was solicited. Available values are: • **true**: The message was solicited. • **false**: The message was unsolicited.
Indications	**Conditional**. • **DoNotCache**: **true**, requests that the application not cache this refresh message. • **ClearCache**: **true**, requests that the application clear the cache. • **Complete**: **true**, indicates that this message completes the refresh.
PartNum	Optional. Specifies the part number of a multi-part refresh.
QoS	Optional. Specifies the quality of service 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.
ItemGroup	Optional. Associates the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies the permission information associated with content on this stream.
extendedHeader	Not used.
ServiceName	**Required**. Specifies the name of the service from which the consumer wants to request the item.
	NOTE: The consumer application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Required**. Specifies the ID of the service that provides the item.
	NOTE: The consumer application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	Optional. **NameType** should match the **NameType** specified in the request. If absent, it is assumed to be **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. **Name** should match the requested name.
Filter	Not used.
Identifier	Not used.

Table 71: Symbol List Refresh Message

COMPONENT	DESCRIPTION / VALUE
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 71: Symbol List Refresh Message (Continued)

11.2.3 Symbol List Update Message

A Symbol List Update Message is encoded using **UpdateMsg** and sent by Open Message Model provider and non-interactive provider applications. It adds or removes items from the list. Updates will not be received before images, and a true snapshot is supported.

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Update response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_SYMBOL_LIST = 10**
Indications	**Conditional**. • **DoNotCache**: **true**, indicates to not cache this update message. • **DoNotConflate**: **true**, indicates to not conflate the update message.
QoS	Optional. Specifies the quality of service at which the stream is provided.
UpdateTypeNum	Not used.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
ConflatedCount	Optional. If a provider sends a conflated update, **ConflatedCount** specifies how many updates are in the conflation. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** is set to **true** in the request.
ConflatedTime	Optional. If a provider sends a conflated update, **ConflatedTime** specifies the time interval (in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting the **ReqMsg.ConflatedInUpdates** is set to **true** in the request.
PermissionData	Optional. Specifies the permission information associated with only the contents of this update.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**. **ServiceName** specifies the name of the service that provides the data. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**. Specifies the ID of the service that provides the item. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. Set this to match the **NameType** in the item's request message (typically **INSTRUMENT_NAME_RIC = 1**). If absent, it is assumed to be **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true**. Specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.

Table 72: Symbol List Update Message

COMPONENT	DESCRIPTION / VALUE
Payload	**Required**. The payload contains a **Map**, where each entry represents an item in the list. Each map entry contains a **FieldList** with additional information about that item.

Table 72: Symbol List Update Message (Continued)

11.2.4 Symbol List Status Message

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

NOTE: The provider should send the **Name** and **ServiceName** only in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then the **Name** and **ServiceName** must be provided for every Status response message.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_SYMBOL_LIST = 10**
State	Optional. Current state information associated with the data and stream.
Indications	**Conditional**. **ClearCache**: **true**, indicates to clear the cache.
QoS	Optional. Specifies the quality of service at which the stream is provided.
ItemGroup	Optional. The provider can use this to change the item's **ItemGroup**.
PermissionData	Optional. Specifies new permissioning information associated with the stream's contents.
extendedHeader	Not used.
ServiceName	**Conditional**. **ServiceName** is required if **MsgKeyInUpdates** was set to **true**. **ServiceName** specifies the name of the service that provides the data. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
ServiceId	**Conditional**. **ServiceId** is required if **MsgKeyInUpdates** was set to **true**. Specifies the ID of the service that provides the item. **NOTE:** The provider application should set either the **ServiceName** or **ServiceId** of the service, but not both.
NameType	**Conditional**. **NameType** is required if **MsgKeyInUpdates** was set to **true**. **NameType** should match the name type specified on the request. If it is not specified, **NameType** defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Conditional**. **Name** is required if **MsgKeyInUpdates** was set to **true**. Specifies the name of the item being provided.
Filter	Not used.
Identifier	Not used.
Attrib	Not used.
Payload	Not used.

Table 73: Symbol List Status Message

11.3 Data: Response Message Payload

The Symbol List payload is a **`Map`**. Each **`MapEntry`** key is an AsciiString symbol. The entry's payload can be empty or contain a **`FieldList`** 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

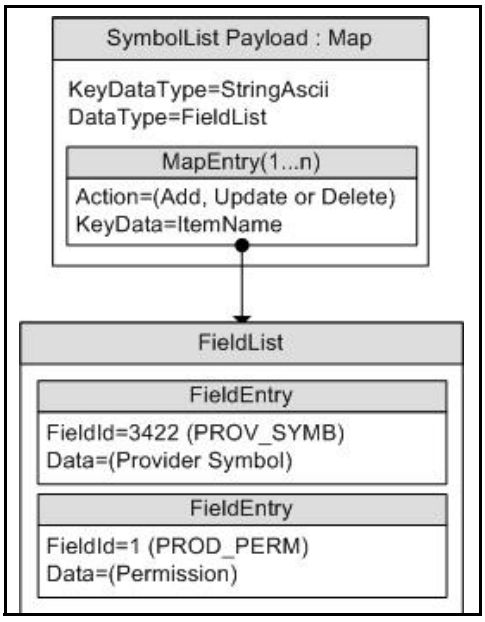


Figure 21. SymbolList Response Message Payload

11.4 Special Semantics

None

11.5 Specific Usage

The payload is a **Map**. No SummaryData is provided. Each Refresh message includes up to 150 **MapEntry**s. DataDefinitions are not used to reduce bandwidth. The **Map.TotalCountHint** is not provided. The **Map.KeyFieldId** is currently not set.

Each **MapEntry**'s key is a Buffer that can be used as a request's **Name** to make a request for an instrument. Each **MapEntry**'s value is a **FieldList** that contains the following information:

- PROV_SYMB (3422): Original symbol provided by the exchange
- PROD_PERM (1): Permission information

The OPRA Venue's SymbolList, **0#OPRA** is a hierarchical SymbolList of SymbolLists. Nested SymbolLists start with **Z#**. For details, refer to the *RDF Direct OPRA Venue Guide*.

Appendix A ReqMsg Payload

A.1 View Definition

The client application can specify interest in a specific subset of fields or elements (known as a 'View'). This is done by encoding an array of the desired fields or elements in the request message payload. The response Message will contain a list of the requested fields or elements and possibly some others depending on factors such as aggregation and the ability of the provider to supply the requested view. Unless otherwise specified, this is supported on any non-administrative Refinitiv Domain Model and any user defined DMM. For more information, refer to the *Enterprise Message API C++ Edition Developers Guide*. When requesting a new view or changing a view, at a minimum, the request message payload contains an element list with the following entries (any default behavior is included in the element's description):

ELEMENT NAME	TYPE	RANGE/EXAMPLES	DESCRIPTION
:ViewType	UInt	1 | 2	**Conditional**. Specifies the content type of the **`:ViewData`** array. Required when specifying a view or when reissuing while wanting to keep the same view. Not required when re-issuing to remove a view. In this case, do not send a payload or view. Available values are: • 1 = VT_FIELD_ID_LIST (this is the default) • 2 = VT_ELEMENT_NAME_LIST
:ViewData	Array of Int or Array of ASCII	An Array of desired entries whose content matches the type as specified by **`:ViewType`**. e.g., a **`:ViewType`** of VT_FIELD_ID_LIST uses an array of field IDs.	**Required**. Field Ids will be encoded as an array of 2 byte fixed length field identifiers. Element names will be variable length Ascii string fields. **`:ViewData`** does not use a default value.

Table 74: View Definition in Payload

A.2 ItemList

The client application can specify interest in multiple items by using a single batch request message. To do this, encode a list of item names in the request message payload. This is supported on any non-administrative Refinitiv Domain Model and any user defined DMM. For further details, refer to *Enterprise Message API C++ Edition Reference Guide.*

For batch request messages, the payload contains, at a minimum, an element list which includes the following element entry:

ELEMENT NAME	TYPE	RANGE/EXAMPLES	DESCRIPTION
:ItemList	Array of ASCII	1 to Array max	**Required**. A list of item names in which the client registers interest. **`:ItemList`** does not use a default value.

Table 75: ItemList in Payload

A.3 Symbol List Behaviors

The client application can specify interest in getting data along with names belonging to the symbol list while requesting a symbol list. By specifying interest in data along with names, a client application does not need to open individual items belonging to symbol list and the items will be opened and data will be provided. To do this, encode a request message payload with an element list that has an element entry which specifies symbol list behavior.

For Symbol List request messages that specify interest in data, refer to Section A.3.1.

For further details, refer to *Enterprise Message API C++ Edition Reference Guide.*

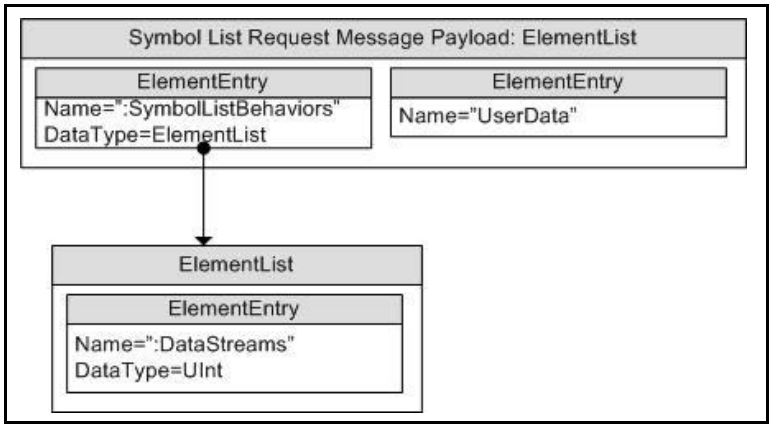


Figure 22. SymbolList Request Message Payload Specifying Symbol List Behavior

A.3.1 Element List Contents

To encode a Symbol List request message that specifies interest in data, include an element list with the following element entry:

ELEMENT NAME	TYPE	DEFAULT	DESCRIPTION
:SymbolListBehaviors	ElementList	ElementList containing ElementEntry of **DataStreams** set to **0**.	Indicates any expected data behavior of individual items that will be opened from the symbol list. If this element is absent, individual streams will not be opened. The contents of **`:SymbolListBehaviors`** are extensible.

Table 76: Request Message Payload for Symbol List Domain Specifying Symbol List Behaviors

A.3.2 Contents of :SymbolListBehaviors

The following is the contents of the **:SymbolListBehaviors** element entry.

ELEMENT NAME	TYPE	RANGE	DESCRIPTION
:DataStreams	UInt	0 - 2	Indicates whether the consumer wants the individual items of the symbol list to be opened as streaming or non-streaming or not opened at all. For more information refer to the *Enterprise Message API C++ Edition Developers Guide*. **:DataStreams** uses the following bit-masks: • **0x0**: The consumer is interested only in getting the names and no data on the individual items of the symbol list. This is the default behavior. • **0x1**: The consumer is interested in getting the individual items of the symbol list opened as streaming. • **0x2**: The consumer is interested in getting the individual items of the symbol list opened as snap-shots.

Table 77: :SymbolListBehaviors ElementEntry Contents

Document ID: EMAC368UMRDM.230
Date of issue: January 2023