ELEKTRON MESSAGE API 3.0.4

REUTERS DOMAIN MODELS (RDM) USAGE GUIDE

C++ EDITION

Document Version: 1.2
Document ID: EMAC304UMRDM.160
Date of Issue: 19 August 2016

THOMSON REUTERS

Contents

Figures

Tables

Chapter 1 Introduction

1.1 About this Manual

The *RDM Usage Guide* describes how the Reuters Domain Models (RDM) are defined in terms of the Open Message Model (OMM). RDM data is available from Reuters Data Feed Direct and an Advanced Distribution Server (ADS, an Enterprise Platform component) using the Elektron Messaging API (EMA).

1.2 Audience

This guide targets software programmers who are familiar with the EMA message model and wish to develop EMA-based applications to access RDM data. It is helpful, but not required, to have an understanding of OMM concepts before using this document.

1.3 Open Message Models (OMM)

The ***OMM*** is a collection of message header and data constructs. Some OMM 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. OMM 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 OMM model, also referred to as a domain model, is described within that model's definition and is not coupled with the API. The OMM is the flexible tool that simply provides the building blocks to design and produce domain models to meet the needs of the system and its users. EMA provides structural representations of the OMM constructs and manages the RWF binary encoded representation of the OMM. EMA users can leverage the provided OMM constructs to consume or provide OMM data throughout the Enterprise Platform.

1.4 Reuters Wire Format (RWF)

Reuters Wire Format (RWF) is the encoded representation of OMM. RWF 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. RWF allows for serializing OMM message and data constructs in an efficient manner while still allowing rich content types. RWF can distribute field identifier-value pair data, self-describing data, as well as more complex, nested hierarchal content.

1.5 Acronyms and Abbreviations

ACRONYM	DEFINITION
ADH	Advanced Data Hub
ADS	Advanced Distribution Server
API	Application Programming Interface
ATS	Advanced Transformation System
DACS	Data Access Control System
DMM	Domain Message Model
IDN	Integrated Data Network
OMM	Open Message Model
QoS	Quality of Service
RDM	Reuters Domain Model
RDF Direct	Reuters Data Feed Direct
RMTES	Reuters Multi-Lingual Text Encoding Standard
RSSL	Reuters Source Sink Library
RWF	Reuters Wire Format
TS1	Time Series One

Table 1: Acronyms and Abbreviations

1.6 References

For additional EMA documentation, refer to *EMA 3.0.4 C++ Edition Developers Guide*.

1.7 Documention Feedback

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

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

1.8 Conventions

1.8.1 Typographic and Syntax

- The Reuters Domain Models (RDMs) are described in terms of OMM concepts.
- Class, methods, and types are shown in `orange, Lucida Concole` text
- The names of the data members correspond to the method names for both get/set from the EMA interface, with the get prefixes removed and the first character always upper case.
- Parameters, filenames, and directories are shown in **Bold** font.
- When included in the bodytext, new concepts are called out in ***Bold, italics*** the first time they are mentioned.
- Document titles are formatted in *italics*.
- Integer constants are defined in all capital letters with underscores (e.g. **MMT_LOGIN = 1**, **SERVICE_INFO_FILTER**). In EMA C++, they can be found in the **thomsonreuters.ema.rdm** namespace and in the **Access/Include/EmaRdm.h** file.
- ***Payload*** generally refers to message payload.
- The names of `filterId` values (e.g. **SERVICE_INFO_ID**) correspond to the flag value enumeration defined for use with message key's `filter` (**SERVICE_INFO_FILTER**). Names maybe shortened for clarity (e.g. **DirectoryInfo**).
- Dot-separated notation indicates data available within a hierarchy. Each period can indicate a class a data member, an entry or an element name.

1.8.2 Definitions

- ***Required*** means the data must be provided or set.
- ***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. The data will 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.

Chapter 2 Domain Model Overview

2.1 Introduction

A ***Domain Message Model (DMM)*** describes a specific arrangement of OMM 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 Reuters Domain Models VS User Defined Model

2.2.1 Reuters Domain Models (RDMs)

An ***RDM*** is a domain message model typically provided or consumed by a Thomson Reuters product, such as the Enterprise Platform Real-Time, Data Feed Direct, or Elektron. Some currently defined RDMs 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. MarketPrice, MarketByOrder, MarketByPrice). Thomson Reuters's defined models have a domain value of less than 128.

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

DOMAIN	PURPOSE
Login	Used to authenticate users and advertise/request features that are not specific to a particular domain. Use and support of this domain is required for all OMM applications. This is considered an administrative domain. Content is required and expected by many Thomson Reuters components, and following the domain model definition is expected. See Chapter 3 for detailed information.
Source Directory	Used to advertise information about available services and their state, quality of service (QoS), and capabilities. This domain is also used to convey any group status and group merge information. Interactive and Non-Interactive OMM Provider applications require support for this domain. Thomson Reuters strongly recommends that OMM Consumers request this domain. This is considered an administrative domain. Content is required and expected by many Thomson Reuters components, and following the domain model definition is expected. See Chapter 4 for detailed information.
Dictionary	Used to provide dictionaries that may be necessary to decode data. Dictionary domain use is optional. It is recommended for Provider applications to support this. This is considered an administrative domain. Content is required and expected by many Thomson Reuters components, and following the domain model definition is expected. See Chapter 5 for detailed information.
MarketPrice	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. See Chapter 6 for detailed information.

DOMAIN	PURPOSE
MarketByOrder	Provides access to Level II full order books. Contains a list of orders, keyed by the order Ids along with the information related to that order, such as price, whether it is a bid/ask order, size, quote time, or market maker identifier. See Chapter 7 for detailed information.
MarketByPrice	Provides access to Level II market depth information. Contains a list of price points (using that price and bid/ask side as its key) along with the information related to that price point. See Chapter 8 for detailed information.
MarketMaker	Provides access to market maker quotes and trade information. Contains a list of market makers (using that market maker's Id as its key) along with the information such as that market maker's bid and asking prices, quote time, and market source. See Chapter 9 for detailed information.
YieldCurve	This section defines a yield curve model as is currently supported by the ATS. The YieldCurve domain shows the relation between the interest rate and the term, or time to maturity, associated with the debt of a borrower. See Chapter 10 for detailed information.
SymbolList	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. See Chapter 11 for detailed information.

Table 2: Reuters Domain Model Overview

2.2.2 User-Defined Domain Model

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

2.2.3 Domain Message Model Creation

The focus of this document is to discuss defined RDMs that are capable of flowing through EMA. OMM allows users of EMA to create their own Domain Message Models (DMMs) in addition to those described in this document. When defining a DMM, several things should be considered:

- Is a new DMM really needed, or can you express the data in terms of an existing RDM?
- The DMM should be well-defined. Following the design templates used in this document may be a good idea. The structure, properties, use cases, and limitations of the DMM should be specified.
- While OMM provides building blocks that can structure the data in many ways, the semantics of the data must abide by the rules of OMM. For example, custom DMMs should follow the request, refresh, status, and update semantics implicitly defined by those messages. More flexible messaging 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 RDMs. The domainType of a custom DMM must be between 128 and 255 (including 128 and 255).
- You might want to work with Thomson Reuters to define a published RDM, rather than using a custom DMM. This ensures the most seamless interoperability with future RDMs and other Thomson Reuters products.

2.3 Message Concepts

The following table describes the mapping of OMM concepts with actual interfaces. For clarity and consistency, the Message concept will be referenced throughout the rest of the *RDM 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 OMM Consumer / OMM Interactive Provider Initial Interaction

An OMM consumer application can establish connections to other OMM interactive provider applications, including the Thomson Reuters Enterprise Platform, Data Feed Direct, and Elektron. 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. MarketPrice, MarketByOrder).

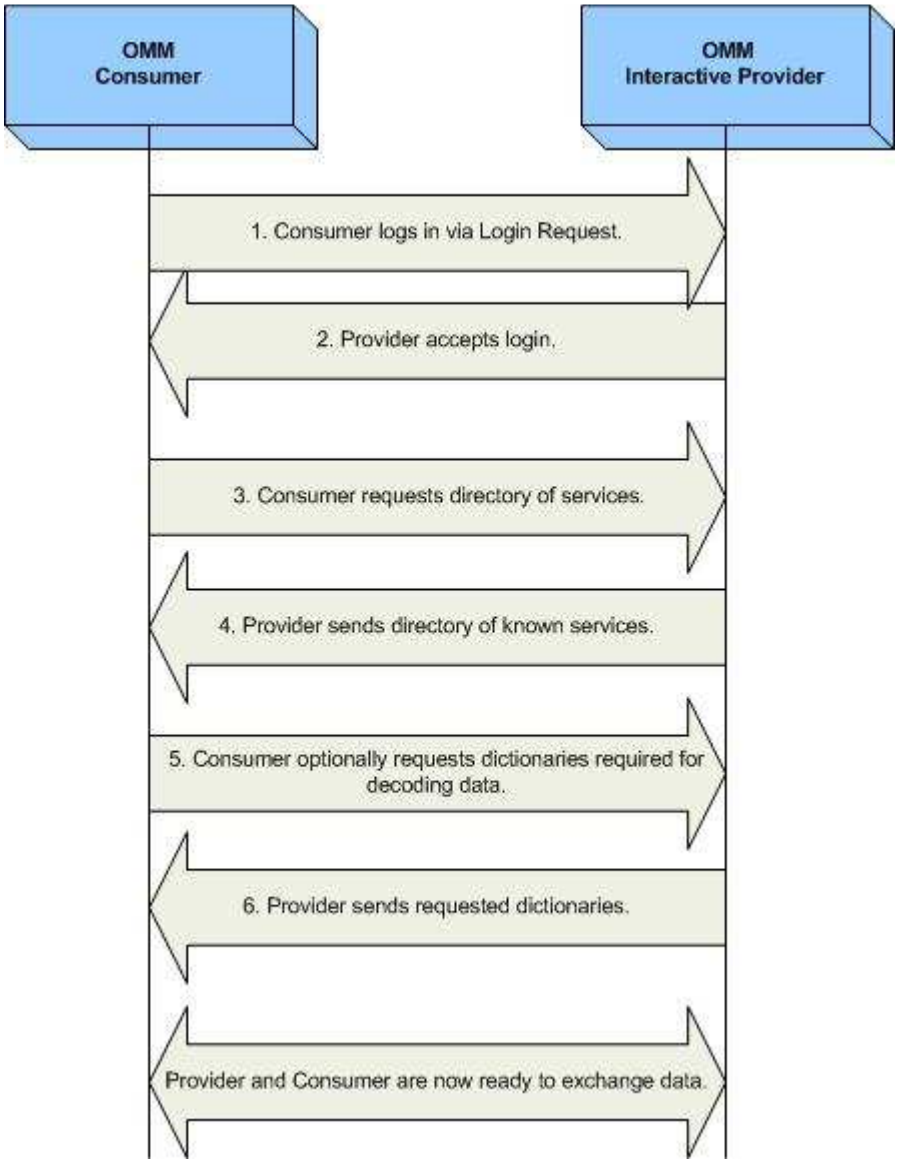

Figure 1: OMM Consumer and Provider Application Initial Interactions

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

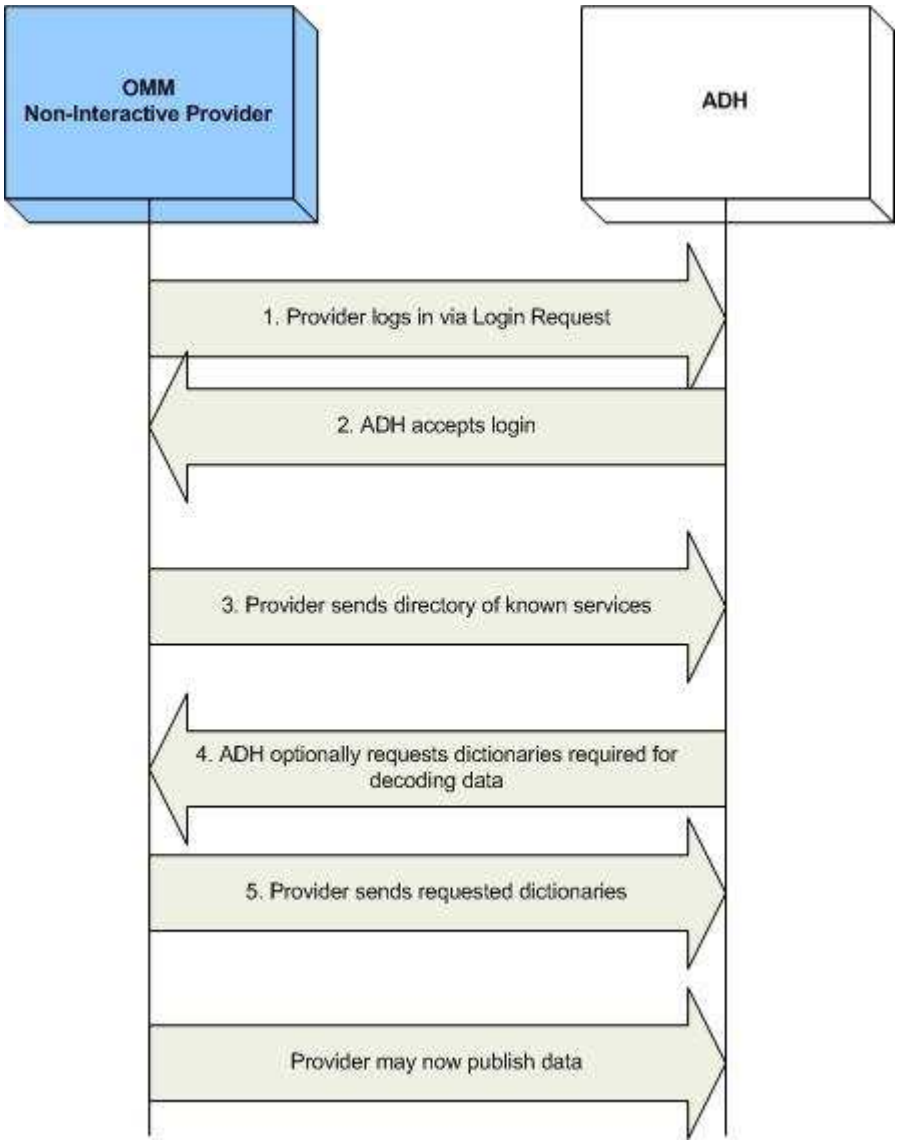

Figure 2: OMM Consumer and ADH Initial Interactions

2.5 Sending and Receiving Content

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

- The consumer sends a request (refer to Section 2.3) message containing the name of an item it is interested in.
- The provider first responds with a refresh (refer to Section 2.3) message to bring the consumer up-to-date with all currently available information.
- As data changes, the provider sends update (refer to Section 2.3) messages (if the consumer requested streaming information).
- When the consumer is no longer interested, it sends a close message to close the stream (or, if the provider needs to close the stream, it uses a status (refer to Section 2.3) message).

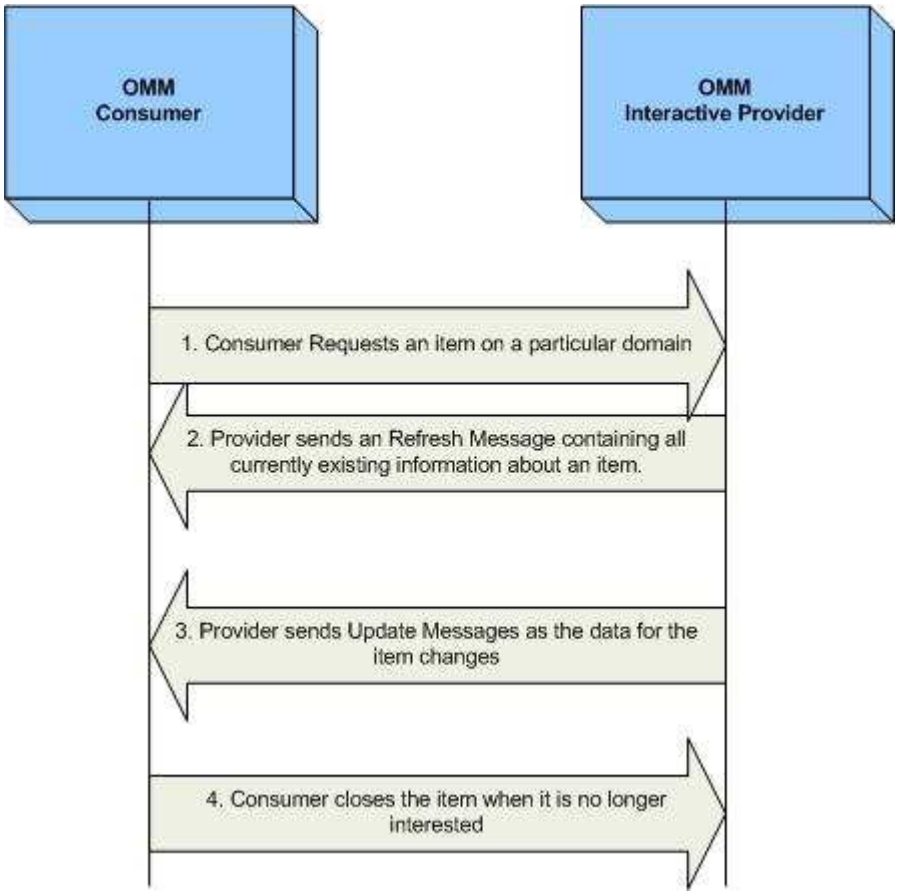

Figure 3: OMM Consumer and Interactive Provider general Domain Use

2.6 General EMA 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 domain's detailed description. This section briefly defines these concepts. Refer to the *EMA 3.0.4 C++ Edition Developers Guide* for further details.

2.6.1 Snapshot and Streaming Requests

Many domains generally support issuing a request (see Section 2.3) message with or without setting `ReqMsg.InterestAfterRefresh`. When the method is passed as **true** value, the request is known as a "streaming" request, meaning that final refresh will be followed by updates.

When a snapshot (non-streaming) request is made, the stream will be closed after the final part of the refresh is received. The final refresh is indicated by setting `RefreshMsg.Complete` with **true** value on the refresh (see Section 2.3) message. The consumer should be prepared to receive status or update messages (see Section 2.3) between the first and final parts of the response refresh (if the domain supplies only a single part of the response 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 subsequent request message (Section 2.2) 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 by setting `ReqMsg.Pause` with **true** value. 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 by calling `ReqMsg.InterestAfterRefresh` with **true** value. 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 *EMA 3.0.4 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, you must call `RefreshMsg.ClearCache` with a value of **true**. For further details on using the clear cache flag, refer to the *EMA C++ Reference Manual*.

```
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 pass **true** value on an unsolicited refresh, the cache is cleared and populated with new data.
  - If this method is not set to **true** value on the unsolicited refresh, new data is overlaid onto the existing data. In this case, the resulting image / refresh will be a superset of the fields currently contained in the 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 flowing 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 3.2.4. For more information on views, refer to the *EMA 3.0.4 C++ Edition Developers Guide.*

However it should be noted that, currently, ADS supports view only on Market Price Level 1 data. As a result, this causes EMA 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 request (refer to Section 2.3) message. 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 via the **SupportBatchRequests** Login attribute, as described in Section 3.2.4. For more information on batch, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

2.6.6 Posting

Posting offers an easy way for an OMM 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 post (see Section 2.2) message to post content to the system. This message can contain any OMM container type as its payload, but this often is an `Msg`. A provider indicates support for posting via the **SupportOMMPost** Login attribute, as described in Section 3.2.4. For more information on posting, refer to *EMA 3.0.4 C++ Edition Developers Guide*.

Chapter 3 Login Domain

3.1 Description

The ***Login*** domain registers a user with the system, after which the user can request[2], or post[3] RDM content. A ***Login*** request may also be used to authenticate a user with the system. A consumer application must log in before it can request or post content.

The following sections detail the use of each message within the Login domain, while a brief summary of Login Authentication is included in Section 3.4.

3.2 Usage

3.2.1 Login Request Message

A Login request message is encoded using `ReqMsg` with default or users configured values and sent by `OmmConsumer` 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 OMM Interactive Provider can use the Login request information to authenticate users with DACS.

Users may configure 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

[2] Consumer applications can request content after logging into the system.

[3] Consumer applications can post content after logging into the system.

The Login request must be streaming (i.e., setting `ReqMsg.InterestAfterRefresh` 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 setting `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. See the *EMA 3.0.4 C++ Edition Developers Guide* for more information.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_LOGIN = 1**
`Interactions`	**Required**. • `InitialImage` **true** : indicates initial image is required • `InterestAfterRefresh` **true** : indicates streaming request is required Optional. • `Pause` **true** : indicates pause is required Streaming request is required before any other requests. A Pause request is a request to pause all item streams associated with the login. Streaming request after a Pause request will resume all item streams associated with the login. Non-streaming request is not supported.
`NameType`	Optional. Possible values: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** If `NameType` is not set, it is assumed to be **USER_NAME** A type of typically corresponds to DACS user name. This can be used to authenticate and permission a user.
`Name`	**Required**. This should be populated with appropriate content corresponding to the `NameType` specification.
`Attrib`	Optional. Typically an ElementList. See Section **3.2.2** for the contents of the ElementList.

Table 5: Login Request Message

3.2.2 Login Request Elements

You can use Login attribute elements to send additional authentication information and user preferences between the components. The passed in data type of `ReqMsg.Attrib` is the `ElementList`. The predefined elements available on a Login Request are shown in the table below.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
AllowSuspectData	UInt	No	1	0, 1	• **1**: Indicates that the consumer application allows `OmmState.Suspect` state. • **0**: Indicates that the consumer application prefers any suspect data result in the stream being closed with `OmmState.ClosedRecover` state. For more information, refer to Section 3.4.6.
ApplicationAuthorizationToken	ASCII	No	None	Sequence of single byte characters from the base36 character set ([0-9][A-Z])	The application authentication token indicates that application behavior was inspected and approved by Thomson Reuters. For more information on obtaining an application authentication token, contact your Thomson Reuters representative.
ApplicationId	ASCII	No	None	1-65535 e.g. “256”	DACS application ID. If the server authenticates with DACS, the consumer application may be required 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 Reuters and will be uniform across all client systems. IDs from 257 to 65535 are available for site-specific use.
ApplicationName	ASCII	No	None	Name of application e.g. “Consumer”	Identifies the application sending the Login request or response message. When present, the application name in the Login request identifies the OMM Consumer, and the application name in the Login response identifies the OMM Provider.
DownloadConnectionConfig	UInt	No	0	0, 1	Request for downloading the configuration. • **1**: Indicates the user wants to download connection configuration information. • **0** (or if absent): Indicates that no connection configuration information is desired.
InstanceId	ASCII	No	None	Any ASCII String, e.g. “Instance1”	**InstanceId** is used to differentiate applications running on the same client host. Because it is set by the client, it does not guarantee uniqueness across different applications on the same client host.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
Password	ASCII	No	None	“my1pword”	Sets the password for logging into the system. This information may be required and encrypted in the future.
Position	ASCII	No	None	ip addr/hostname ip addr/net e.g.”192.168.1.1/net”	DACS position. If the server is authenticating with DACS, the consumer application might be required to pass in a valid position.
ProvidePermissionExpressions	UInt	No	1	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.
ProvidePermissionProfile	UInt	No	1	0, 1	When specified on a Login Request, indicates that a consumer desires the permission profile. The permission profile can be used by an application to perform proxy permissioning.
SupportProviderDictionaryDownload	UInt	No	0	0, 1	Indicates whether the server supports the Provider Dictionary Download: • **1**: The server supports the Provider Dictionary Download. • **0**: The server does not support the Provider Dictionary Download feature. If this element is not present, the server does not support the Provider Dictionary Download feature. For more information on Provider Dictionary Download, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.
Role	UInt	No	0	**LOGIN_ROLE_CONS = 0,** **LOGIN_ROLE_PROV = 1**	Indicates the role of the application logging onto the system. • An OMM consumer application should specify its role as **LOGIN_ROLE_CONS**. • EMA will default role element with value of 1. OMM consumer applications typically connect to a different port number than non-interactive provider applications. Role information allows the Enterprise Platform to detect and inform users of incorrect port use.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
SingleOpen	UInt	No	1	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 3.4.6.
DisableDataConversion	N/A	N/A	N/A	N/A	Reserved by TR.

Table 6: Login Request Attribute Elements

3.2.3 Login Refresh Message

A Login refresh message is encoded using `RefreshMsg` and sent by OMM interactive provider applications. This message is used to respond to a Login Request message after the user's Login is accepted. An OMM Provider can use the Login request information to authenticate users with DACS. 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.

The content of a Login Refresh message is expected to be atomic and contained in a single part, therefore `RefreshMsg.Complete` with **true** value should be set. If the login refresh is sent in response to a request, the `RefreshMsg.Solicited` with **true** value should be set to indicate that this is a solicited response.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required.** **MMT_LOGIN = 1**
`State`	Optional. For Refresh, when accepting Login: • `StreamState` == `OmmState.Open` • `DataState` == `OmmState.Ok` • `StatusCode` == `OmmState.None`
`Solicited`	**Required.** • **true**: indicates refresh solicited • **false**: indicates refresh unsolicited
`Indications`	**Required**. • `Complete` **true**: indicates refresh complete Optional. • `ClearCache` **true**: indicates clear cache
`NameType`	Optional. Possible values: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** If `NameType` is not set, it is assumed to be `NameType` of **USER_NAME**. If present, the value should match the type specified in the Login request.
`Name`	Optional. If `Name` is set, it should match the `Name` specified in the Login request and contain appropriate content corresponding to the `NameType` specification.
`Attrib`	Optional. Typically an `ElementList`. For the contents of the `ElementList`, refer to Section 3.2.4.
`Payload`	Optional. Typically present when login requests connection configuration or permission profile information. This is sent as an ElementList. For payload details, refer to Section 3.3.1.

Table 7: Login Refresh Message

3.2.4 Login Refresh Elements

The Login attribute elements can be used to send additional authentication information and user preferences between components. The Data is the ElementList. The following table includes predefined elements available on a Login Refresh.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
AllowSuspectData	UInt	No	1	0, 1	• **1**: The provider application passes along `OmmState.Suspect` state. • **0**: The provider application does not pass along `OmmState.Suspect` state. Any suspect stream will be closed with a `OmmState.ClosedRecover` state. For more information, refer to Section 3.4.6.
ApplicationId	ASCII	No	None	1-65535 e.g. “256”	DACS application ID. If the server is authenticating with DACS, the consumer application may be required to pass in a valid **ApplicationId**. This should match what was sent in the request. This must be unique for each application. IDs from 1 to 256 are reserved for permanent market data applications. These are assigned by Reuters and will be uniform across all client systems. IDs from 257 to 65535 are available for site-specific use.
ApplicationName	ASCII	No	None	name of application e.g. “EMA”	Identifies the application sending the Login request or response message. When present, the application name in the Login request identifies the OMM Consumer and the application name in the Login response identifies the OMM Provider.
Position	ASCII	No	None	ip addr/hostname ip addr/net e.g.”192.168.1.1/net”	DACS position. If the server authenticates with DACS, 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.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
ProvidePermissionExpressions	UInt	No	1	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.
ProvidePermissionProfile	UInt	No	1	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.
SingleOpen	UInt	No	1	0, 1	• **1**: The provider drives stream recovery. • **0**: The provider does not drive stream recovery; it is the responsibility of the downstream application. For more information, refer to Section 3.4.6.
SupportBatchRequests	UInt	No	0	0-7	Indicates whether the provider supports a batch request. Batch requesting allows a consumer to specify multiple items, all with matching attributes, in the same request message. For more information on batch requesting, refer to the *EMA 3.0.4 C++ Edition Developers Guide*. These are bit-masks: • **0x0**(or if absent): The provider does not support batch requesting. • **0x1**: The provider supports batch requesting. • **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.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
SupportOMMPost	UInt	No	0	0, 1	Indicates whether the provider supports OMM Posting: • **1**: The provider supports OMM Posting and the user is permissioned. • **0**: The provider supports the OMM Post feature, but the user is not permissioned. • If this element is not present, then the server does not support the OMM Post feature. For more information on posting, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.
SupportPauseResume	UInt	No	0	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.
SupportProviderDictionaryDownload	UInt	No	0	0, 1	Indicates whether the server supports the Provider Dictionary Download: • **1**: The server supports the Provider Dictionary Download. • **0**: The server does not support the Provider Dictionary Download feature. If this element is not present, the server does not support the Provider Dictionary Download feature. For more information on Provider Dictionary Download, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.
SupportOptimizedPauseResume	UInt	No	0	0, 1, or other positive integers greater than 0	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 via a pause of the Login stream. For more information on Pause and Resume, refer to the *EMA 3.0.4 C++ Edition Developers Guide*. • **1** (or other positive integers greater than 0): The server supports optimized pause and resume. • **0** (or if absent): The server does not support optimized pause and resume.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
SupportViewRequests	UInt	No	0	0, 1	Indicates whether the provider supports requesting with Dynamic View information. A Dynamic View allows a user to request only the specific contents of the response information in which they are interested. For more information on Dynamic View use, refer to the *EMA 3.0.4 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.
SupportStandby	UInt	No	0	0, 1	Indicates whether the provider supports Warm Standby functionaltity. If supported, a provider to run as either 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. When informed of an Active's failure, the Standby begins sending responses and assumes Active functionality. • **1**: The provider can support a role of Active or Standby in a Warm Standby group. • **0** (or if absent): The provider does not support warm standby functionality.
SupportEnhancedSymbolList	UInt	No	0	0, 1	Indicates whether the provider supports the enhanced symbol list functionality. • **1**: The provider supports data streams along with names. • **0** (or if absent): The provider does not support enhanced symbol list functionality. The provider sends only names and will not send data.

Table 8: Login Refresh Attribute Elements

3.2.5 Login Status Message

OMM Provider and OMM non-interactive provider applications use the Login status message to convey state information associated with a stream. Such state information can indicate that a stream cannot be established or inform a consumer of a state change associated with an open 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 user needs to retry login, for example when DACS is not yet connected to ADS: • StreamState == OmmState.ClosedRecover • DataState == OmmState.Suspect • StatusCode == OmmState.NotAuthorized
SeqNum	Optional.
NameType	Optional. Possible values: • **USER_NAME** • **USER_EMAIL_ADDRESS** • **USER_TOKEN** If NameType is not set, it is assumed to be NameType of **USER_NAME** If present, the value should match the type specified in the ***Login*** request.
Name	Optional. If present, it should match the Name specified in the ***Login*** request and should contain appropriate content corresponding to the NameType specification.
Attrib	Optional. Typically an ElementList. See Section 3.2.4 for the contents of the ElementList.

Table 9: Login Status Message

3.2.6 Login Update Message

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

3.2.7 Login Close Message

A consumer can close the login stream by calling an `OmmConsumer` destructor. Closing a login stream is equivalent to a "Close All" type of message, where all open streams are closed (thus all the streams associated with the user are closed). A provider can log off a user and close all the user's streams via a ***Login Status*** message see Section 3.2.5.

3.2.8 Login Generic Message

Note: Generic Message(s) (refer to Section 2.3) are currently supported for LOGIN Reuters Domain Model only for the sending and receiving of information related to "ConsumerConnectionStatus" Warm Standby Mode.

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

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

Content for a Login Generic message is expected to be atomic and contained in a single part, therefore `GenericMsg.Complete` should be set to **true**.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required.** **MMT_LOGIN = 1**
`Indications`	**Required.** • `Complete` **true**: indicates refresh complete
`Name`	**Required**. Name must be set to "ConsumerConnectionStatus."
`Payload`	**Required**. Indicates whether a provider acts as an Active or a Standby server. This is sent as a `Map` type. For further details, refer to Section 3.3.2.

Table 10: Login Generic Message

3.2.9 Login Post Message

OMM consumer applications can encode and send Post messages on their Login stream. Any item can be posted to via the login stream. This is known as ***off-stream posting*** because items are being posted without using that item's dedicated stream. Posting on an item's own, dedicated stream is referred to as ***on-stream posting***.

When an application is posting off-stream (i.e., on the Login stream), the `PostMsg` requires both `Name` and `ServiceId` information. For more details on posting, see the *EMA 3.0.4 C++ Edition Developers Guide*.

3.2.10 Login Ack Message

OMM Provider applications encode and send Ack messages to acknowledge receipt of Post messages. This message is used when a consumer is posting off-stream and requests an acknowledgment. The acknowledgement contains a positive (ACK) or negative (NACK) code. For more details on posting, see the *EMA 3.0.4 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 as the payload of refresh (see Section 2.3) message. This information can be useful for load balancing connections across multiple providers or ADS components.

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

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

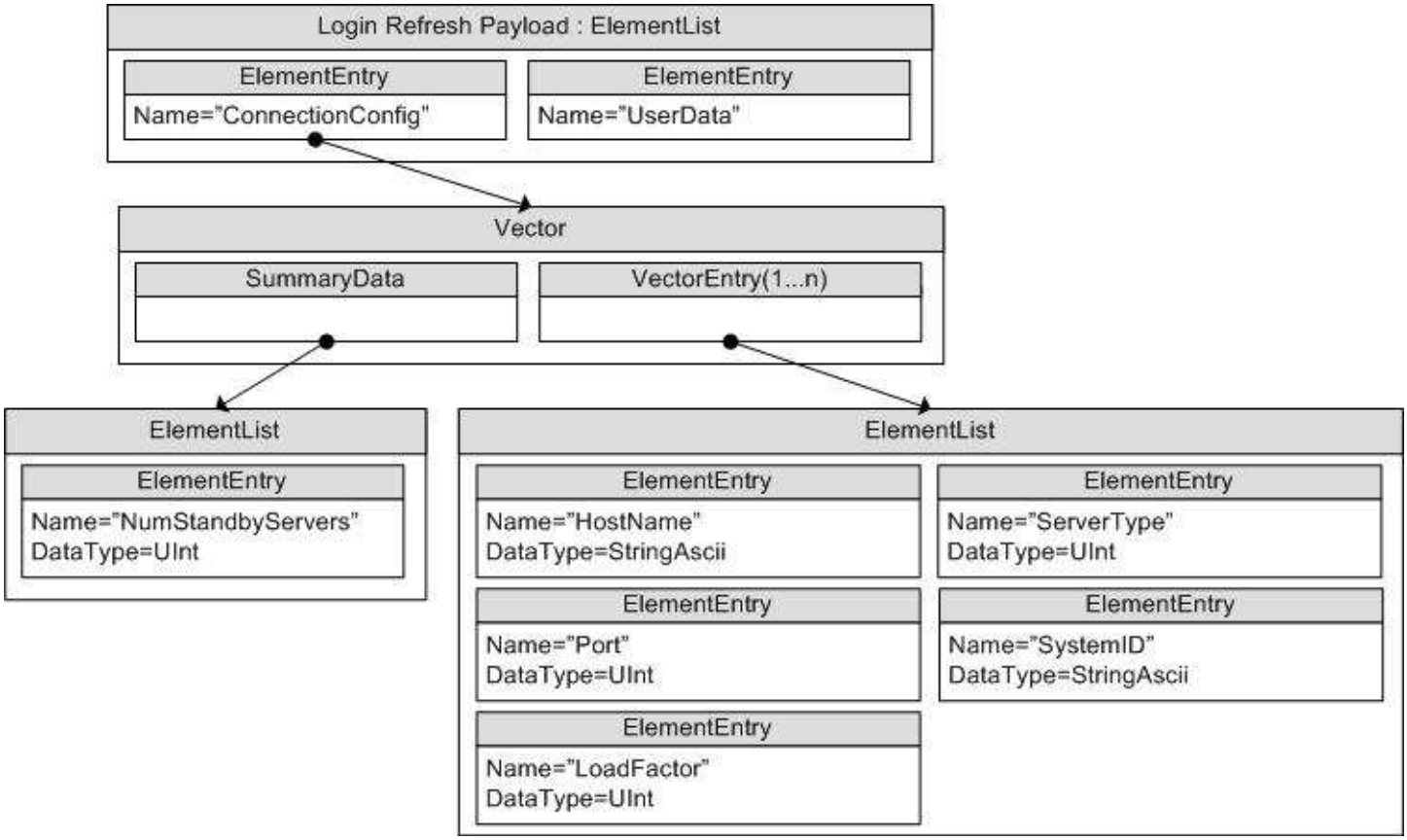

Figure 4: Login Refresh Message Payload

When present, the summary data `ElementList` contains the following element:

NAME	TYPE	DEFAULT	REQUIRED	RANGE / EXAMPLE	DESCRIPTION
NumStandbyServers	UInt	None	Yes	0 - MAXINT value	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 11: `Vector` SummaryData ElementList Contents

Each `VectorEntry` contains an `ElementList`, each list describing a single provider. Possible elements in this list are as follows:

NAME	TYPE	DEFAULT	REQUIRED	RANGE / EXAMPLE	DESCRIPTION
Hostname	ASCII	None	Yes, when payload is present.	“myHostName” “192.168.1.100”	The potential provider’s IP address or hostname.
Port	UInt	None	Yes, when payload is present.	14002	The potential provider’s port number.
LoadFactor	UInt	65535	No	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.
ServerType	UInt	1 (Standby)	No	0 (Active), 1 (Standby)	The providers expected behavior, when warm standby is employed: • **0**: This provider is expected to be used as an Active server. • **1**: This provider is expected to be used as a Standby server. All other values are reserved for future use.
SystemID	ASCII	None	No		For future use.

Table 12: `ElementList` Contents

3.3.2 Login Generic Message Payload

The Login Generic (Section 2.3) Message `Payload` is formatted as a `Map` type, with a `KeyDataType` of **ASCII** and a `DataType` of **ElementList**. Each keyData is a `ServiceName`. Each `ElementList` contains one `ElementEntry`. There is no summary data and typically only one map entry that inform the provider of its warm standby role. Specific information is contained in Figure 5 and Table 13.

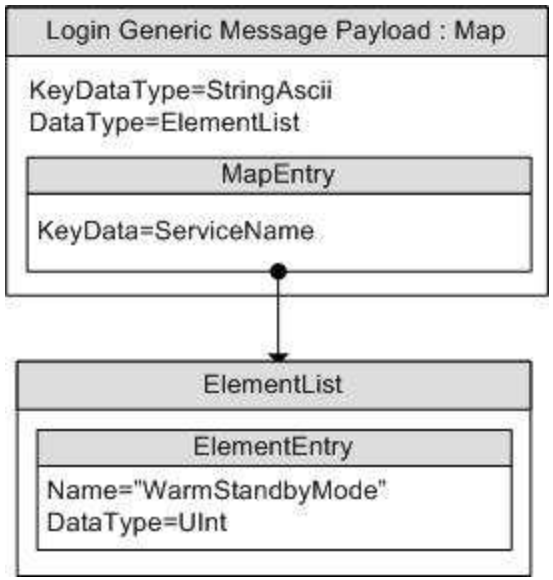

Figure 5: Login Generic Message Payload

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
WarmStandbyMode	UInt	Yes	None	0: Active; 1: Standby	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.

Table 13: `MapEntry`'s Elements

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 EMA Consumer sends a login request to an OMM 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 Authentication

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

- If the Login is accepted, the provider should send a Refresh message with `StreamState` = `OmmState.Open`, `DataState` = `OmmState.Ok`, and `StatusCode` = `OmmState.None`.
- A login can be rejected or closed after it was originally accepted by sending a Status message with `StreamState` = `OmmState.Closed` (or `OmmState.ClosedRecover` if the user is allowed to attempt another login), and `DataState` = `OmmState.Suspect`.
- If provider closes the login stream, all other streams related to that login are implicitly closed without sending a any item Status messages to the consumer.
- If the consumer application calls the `OmmConsumer` destructor, all streams related to that login of `OmmConsumer` are automatically closed without sending any item Close messages to the provider or a Status messages back to the consumer.
- If login stream is not open by EMA or it was closed by either the consumer or the provider, the consumer must create a new `OmmConsumer` before sending other Request messages.

3.4.4 Multiple Logins

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

3.4.5 Group and Service Status

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

3.4.6 Single Open and Allow Suspect Data

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 specific **SingleOpen** and **AllowSuspectData** settings. If any **SingleOpen** and **AllowSuspectData** configuration causes a behavior contradiction (e.g., **SingleOpen** indicates 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, see the *EMA 3.0.4 C++ Edition Developers Guide*.

Note: 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.

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
`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT
`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT	`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT
New item request when: `AcceptingRequests`= TRUE `ServiceState`= DOWN	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= OPEN `DataState`= SUSPECT	`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT	`StreamState`= CLOSED_RECOVER `DataState`= SUSPECT

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
New item request when: AcceptingRequests= TRUE ServiceState= UP	StreamState= OPEN DataState= OK or SUSPECT based on individual item's state	StreamState= OPEN DataState= OK or SUSPECT based on individual item's state	StreamState= OPEN DataState= OK or SUSPECT based on individual item's state	If DataState == OK: StreamState= OPEN If DataState == SUSPECT: StreamState= CLOSED_RECOVER
New item request when: AcceptingRequests= FALSE ServiceState= UP or DOWN	StreamState= OPEN DataState= SUSPECT based on individual item's state	StreamState= OPEN DataState= SUSPECT based on individual item's state	StreamState= CLOSED_RECOVER DataState= SUSPECT based on individual item's state	StreamState= CLOSED_RECOVER DataState= SUSPECT
Connection goes down	StreamState= OPEN DataState= SUSPECT based on individual item's state	StreamState= OPEN DataState= SUSPECT based on individual item's state	StreamState= CLOSED_RECOVER DataState= SUSPECT based on individual item's state	StreamState= CLOSED_RECOVER DataState= SUSPECT

Table 14: SingleOpen and AllowSuspectData Handling

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** (1).

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

The request and response message `Payload` have no data.

RDF Direct only supports one Login per connection.

3.6 Specific Usage: Enterprise Platform

When sending a Login to a DACS enabled Enterprise Platform, the `Name` should be a valid username in DACS. The `Attrib` containing **ApplicationId** and **Position** will also be used for DACS authorization. The Name is used for scoping some Enterprise Platform configuration. The `NameType` must be **USER_NAME** (1).

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.

The request and response message `Payload` have no data.

The Enterprise Platform only supports one Login per connection.

3.7 Specific Usage: Login Credentials Update Feature

Internally EMA 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, EMA 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, EMA will apply the standard login reissue processing. If no new/updated login credentials are specified by application, during the connection recovery phase EMA will use the previously used ones.

Chapter 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 QoS, and any item group information associated with the service. Each service is associated with a particular `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 Sections 4.3.1.2 and 4.3.1.3.
- Source Mirroring information between an ADH and OMM interactive provider applications. EMA conveys this information 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 use a filter to indicate the specific source-related information in which it is interested. 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 *EMA 3.0.4 C++ Edition Developers Guide*.

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

Thomson Reuters 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` information for all available services. When an appropriate service is discovered by the OMM 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. Such 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 NonStreaming request are supported. If you need the ConsumerStatus generic messages to be sent over the Directory stream, you must specify the interaction as "Streaming" in the request.
Filter	**Required**. A filter indicates the specific information in which a consumer is interested. Available categories include: • **SERVICE_INFO_FILTER** • **SERVICE_STATE_FILTER** • **SERVICE_GROUP_FILTER** • **SERVICE_LOAD_FILTER** • **SERVICE_DATA_FILTER** • **SERVICE_LINK_FILTER** For details on the contents of each filter entry, refer to section 4.3.1.1.
ServiceName	Optional. • If present, the directory request is for this specified service. • If neither the ServiceId nor the ServiceName is specified, then the request is for the entire directory. • If the application intends to request a specific service, then it should set either the ServiceId or ServiceName of the service, but not both.
ServiceId	Optional. • If present, the directory request is for this specified service. • If neither the ServiceId nor the ServiceName is specified, then the request is for the entire directory. • If the application intends to request a specific service, then it should set either the ServiceId or ServiceName of the service, but not both.

Table 15: Source Directory Request Message

4.2.2 Source Directory Refresh Message

A Directory Refresh Message is encoded using a `RefreshMsg` and sent by OMM Provider and OMM 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**.
`Solicited`	**Required.** • **true**: Indicates refresh solicited • **false**: Indicates refresh unsolicited
`Indications`	Optional: • `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 FilterEntries in Table 21.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
`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 Section 4.2.1.
`Payload`	**Required**. The payload contains information about available services in the form of a `Map` where each `MapEntry.Key` is one `ServiceName`. For additional details, refer to Section 4.3.2.

Table 16: Source Directory Refresh Message

4.2.3 Source Directory Update Message

A Source Directory Update Message is encoded using an `UpdateMsg` and sent by OMM Provider and OMM Non-Interactive Provider applications. The 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 *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_DIRECTORY = 4**
`Indications`	Optional: • `Complete` **true**: Indicates refresh complete • `ClearCache` **true**: Indicates a clear cache • `DoNotCache` **true**: Indicates this refresh message must not be cached. • `DoNotConflate` **true**: Indicates this update must not be conflated. For more details, refer to FilterEntries (Table 21).
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
`Filter`	Optional. The `Filter` represents the filter entries being provided in this response. For an update, this conveys only the ID values associated with filter entries present in the update payload. For additional details, refer to the `Filter` member in Section 4.2.1.
`Payload`	**Required**. The payload is `Map` contains only the changed information associated with the provided services. For additional details, refer to Section 4.3.1.

Table 17: Source Directory Update Message

4.2.4 Source Directory Status Message

A Source Directory Status message is encoded using a StatusMsg and sent by OMM provider and OMM non-interactive provider to convey state information associated with the 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 directory stream. • StreamState • DataState • StatusCode
Indications	Optional: • ClearCache **true**: Indicates the application should clear its cache For more details, refer to FilterEntries (Table 21).

Table 18: Source Directory Status Message

4.2.5 Source Directory Generic Message

A Source Directory Generic message is encoded and sent by an ADH when using a 'hot standby' configuration. When running in hot standby mode, the ADH can leverage source mirroring and use a generic message to convey usage information to upstream providers. A generic message can inform providers whether the ADH 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 (see Section 4.3.1.1).

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DIRECTORY = 4**
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages within this stream.
Name	**Required**. As "ConsumerStatus"
Payload	**Required**. The payload is Map whose entries contain the SourceMirroring status for each service. For additional details, refer to section 4.3.1.

Table 19: 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 21.

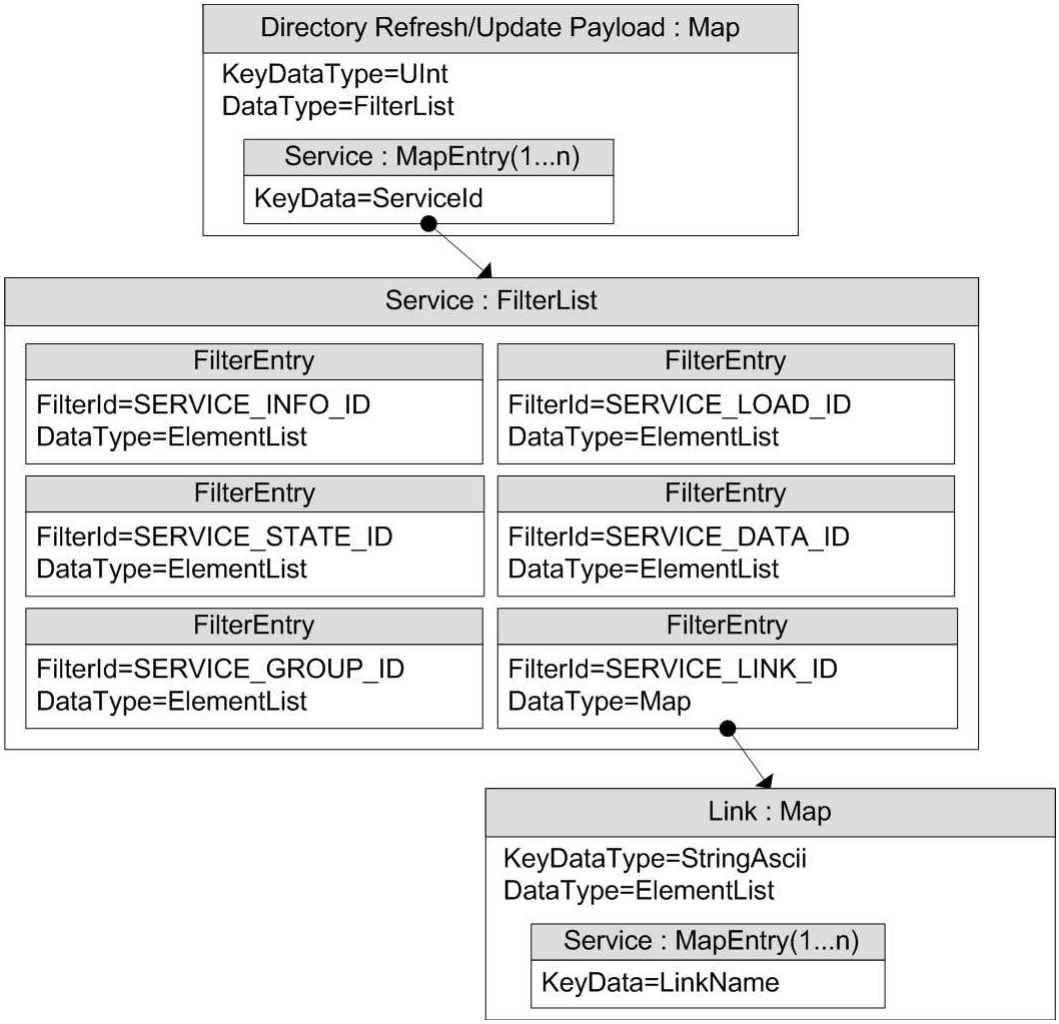

Figure 6: Source Directory Refresh/Update Message Payload

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 20: Source Directory `Map`

There are six categories about a service, each respresent by one `FilterEntry`. Categories can be added or updated in update messages (note that 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.

FILTERENTRY ID (CORRESPONDING FITLTER BIT-VALUE)	TYPE	PERMISSION DATA	PROVIDER SUPPORT REQUIRED	EXTENSIBLE CONTENTS	DESCRIPTION
SERVICE_INFO_ID (SERVICE_INFO_FILTER)	ElementList	Not used	Yes	Yes	Identifies a service and its available data. For details, refer to Section 4.3.1.1.
SERVICE_STATE_ID (SERVICE_STATE_FILTER)	ElementList	Not used	Yes	No	Describe current state of a service (i.e., the service's current ability to provide data). May also be used to change the status of all items associated with this service. In updated response message, the `DoNotConflate` should be set to **true**. For details, refer to Table 23.
SERVICE_GROUP_ID (SERVICE_GROUP_FILTER)	ElementList	Not used	No	No	Manages group information. May be used to change the status of a group of items or merge items from one group to another. Group `FilterEntry` are only sent in updated response message with `DoNotCacheFlag` and `DoNotConflate` shold be set to **true**. For details, refer to Table 24.
SERVICE_LOAD_ID (SERVICE_LOAD_FILTER)	ElementList	Not used	No	Yes	Statistics about how many concurrent streaming requests the service can support and how many it is currently servicing. In the updated ***response*** message, the `DoNotConflate` can optionally be set to **true**. For details, refer to Table 25.
SERVICE_DATA_ID (SERVICE_DATA_FILTER)	ElementList	Not used	No	Yes	Includes broadcast data that applies to all items requested from that service. This information is typically provided in a dedicated update response message and sent independently of other filter entries. The data filter is commonly used with ANSI Page-based data. `DoNotCache` and `DoNotConflate` can optionally be set to **true** to prevent conflation and caching of this content. For details, refer to Table 26.

FILTERENTRY ID (CORRESPONDING FITLTER BIT-VALUE)	TYPE	PERMISSION DATA	PROVIDER SUPPORT REQUIRED	EXTENSIBLE CONTENTS	DESCRIPTION
SERVICE_LINK_ID (SERVICE_LINK_FILTER)	Map	Not used	No	No	Provides information about individual upstream sources that provide data for this service. This is primarily used by systems that aggregate sources (such as the AUH) for identification and load balancing, and is not required to be processed by a consumer application. For details, refer to Table 27.

Table 21: Source Directory MapEntry's FilterEntries

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. MarketPrice, MarketByOrder), available QoS, and the names of any dictionaries needed 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.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
Name	ASCII	Yes	n/a	e.g. IDN_RDF	Service name. This will match the concrete service name or the service group name that is in the `Map.Key`.
Vendor	StringASCII	No	None	e.g. Thomson Reuters	Name of the vendor that provides the data for this service
IsSource	UInt	No	0	0, 1	• **1**: Service is provided directly by original publisher • **0**: Service is a consolidation of multiple sources into a single service.
Capabilities	Array of UInt	Yes	None	e.g. [5, 6]	List of MessageModelTypes which this service can provide. The UInt MesageModelType is extensible, using values defined in the RDM Usage Guide (1-255). For example, a list containing **MMT_DICTIONARY**(**5**) and **MMT_MARKET_PRICE**(**6**) indicates consumer can request dictionaries and market price data from this service.
DictionariesProvided	Array of Ascii	No	None	e.g. RWFFld	List of Dictionary names which this service can provide. A consumer may obtain these dictionaries by requesting them by name on the **MMT_DICTIONARY** MessageModelTypes. For details, refer to Chapter 5.
DictionariesUsed	Array of ASCII	No (Yes if any are used.)	None	e.g. RWFFld, RWFEnum	List of Dictionary names that may be required to process all of the data from this service. Whether or not the dictionary is required depends on the needs of the consumer. For example, if the consumer application is not a display application, it might not need an Enumerated Types Dictionary. For details, refer to Chapter 5.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
QoS	Array of QoS	No	Real-time, TickByTick	e.g . Real-time, TickByTick	The available Qualities of Service. • 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. • Note the default. • If Qos is not provided, EMA assumes the service provides Realtime, Tick-By-Tick data. See the *EMA 3.0.4 C++ Edition Developers Guide* for more information about Qos use and handling.
SupportsQoSRange	UInt	No	0	0, 1	Indicates whether the provider support QoS range when requesting an item. If supported, a consumer may indicate an acceptable range via `ReqMsg.Qos` • **0**: Not supported • **1**: Supported
ItemList	ASCII	No	None		Name of SymbolList that includes all the items that publisher currently provides. If it is not present, this feature is not supported. The consumer requests this item via **MMT_SYMBOL_LIST** MessageModelTypes (See Chapter 11).
SupportsOutOfBandSnapshots	UInt	No	1	0, 1	• **1**: Yes • **0**: No Indicates whether Snapshot requests can be made when the **OpenLimit** has been reached (refer to Section 4.3.1.4).
AcceptingConsumerStatus	UInt	No	1	0, 1	• **1**: Yes • **0**: No Indicates whether a service can accept and process messages related to Source Mirroring (refer to Section 4.2.5).

Table 22: Source Directory Info `FilterEntry`'s Elements

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.

This category should be present in the initial refresh and updated as needed.

The Status element can change the state of items provided by this service. Prior to changing a service status, Thomson Reuters recommends that you issue item or group status messages to update item states.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
ServiceState	UInt	Yes	n/a	0, 1	• **1**: Service is Up • **0**: Service is Down 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. Refer to Section 4.4.2.
AcceptingRequests	UInt	No	1	0, 1	• **1**: Yes • **0**: No Indicates whether the immediate provider can accept new requests and/or handle reissue requests on already open streams. Existing streams remain unaffected, however new request will be rejected. Refer to Section 4.4.2.
Status	State	No	Open (1), Ok (1), None (0), “”	e.g., OmmState.Open OmmSate.Ok, OmmState.None , “OK”	Specifies a status change to apply to all items provided by this service. This status only applies to item streams that have received a refresh or status of **OPEN/OK**. Refer to Section 4.4.3.1 for more details.

Table 23: Source Directory State `FilterEntry`'s 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 refresh message or status message. 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 *EMA 3.0.4 C++ Edition Developers Guide*.

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

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
Group	Buffer	Yes	n/a	e.g., 1.26.102	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 next to each other). The example provided in **Range/Example** column of this table shows such a series, with inserted to help indicate two-byte value.
MergedToGroup	Buffer	No	n/a	e.g., 1.26.110	Used to merge all of the items from one group into another group.
Status	State	No	n/a	e.g.: OmmState.Open OmmState.Ok, OmmState.None , "OK"	A status change to apply to all items whose `ItemGroup` matches the **Group** element. • If you need to convey group status text or code information without changing data status, use the value • If present in the same message as a **MergedToGroup** element, this change should be applied before merge. This change only applies to item streams that have received a refresh or status with a state of **OPEN/OK**. See section 4.4.3.2 for more details.

Table 24: Source Directory Group FilterEntry's 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 mutliple services can provide the desired data, the consumer may use this information to help decide which to use.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
OpenLimit	UInt	No	none	0-MAXUINT	Maximum number of streaming items that the client is allowed to open to this service.
OpenWindow	UInt	No	none	0-MAXUINT	Maximum number of outstanding new refresh requests (i.e. requests for items which are not already open) service can receive at any given time. If **OpenWindow** is **0**, the behavior is the same as setting **AcceptingRequests** to **0** and no open item request will be accepted. The provider should not assume that the OpenWindow becomes effective immediately.
LoadFactor	UInt	No	none	0-65535	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, or something else). 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 higher **LoadFactor** than one that is less loaded.

Table 25: Source Directory Load FilterEntry's 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.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
Type	UInt	No (Yes if Data is present.)	none	• Time(1) • Alert(2) • Headline(3) • Status(4) • Reserved values : 0 – 1023	Explains the content of the Data.
Data	Any Data Type	No	none		Data that should be applied to all items from the service; commonly ised 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 26: Source Directory Data `FilterEntry`'s 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` respresents 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 maily informational. The consumer should use the `State` category to make programmatic decisions about service availability and status.

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
Type	UInt	No	1 (Interactive)	1-2	Indicates whether the upstream source is interactive or broadcast. This does not Describe whether the service itself is interactive or broadcast. • **1**: Interactive • **2**: Broadcast
LinkState	UInt	Yes	n/a	0, 1	Indicates whether the upstream source is up or down. • **0**: Link is Down • **1**: Link is Up
LinkCode	UInt	No	0 (None)	0-3	Provides additional information about upstream source. • **0**: None • **1**: Ok • **2**: RecoveryStarted • **3**: RecoveryCompleted
Text	ASCII	No	“”	n/a	Explains the `LinkState` and `LinkCode`.

Table 27: Source Directory Link `FilterEntry` `Map`'s Entries

4.3.2 Source Directory ConsumerStatus Generic Message Payload

Note: **Generic Message**(s) are supported for the **DIRECTORY** RDM only for sending / receiving information related to ConsumerStatus/Source Mirroring Mode.

The directory data structure for Generic message payload is `Map`. Each MapEntry sends status to one service and is uniquely identified by `ServiceId` (key Data). Each entry contains an `ElementList` with one `ElementEntry` indicating how the provider is used.

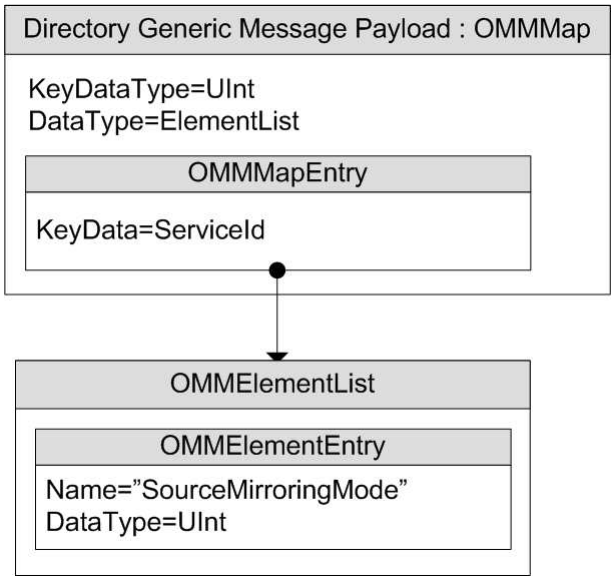

Figure 7: Source Directory Generic Message Payload

ELEMENT NAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLE	DESCRIPTION
SourceMirroringMode	UInt	Yes	none	0 - 2	Indicates how the downstream component is using the service: • **0**: **ActiveNoStandby**. The downstream device is using the data from this service, and is not receiving it from any other service. • **1**: **ActiveWithStandby**. The downstream device is using the data from this service, but is also getting it from another service. • **2**: **Standby**. The downstream device is getting data from this service, but is actually using data from another service. (No reply from the provider application is needed as this is informational only.)

Table 28: Source Directory Generic Message `MapEntry`'s Elements

4.4 Special Semantics

4.4.1 Multiple Streams

Unlike other MessageModelTypes, it is permitted to open two Directory streams with identical message key information. It is also permissible to change the Filter of an open stream.

4.4.2 ServiceState and AcceptingRequests

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

- **ServiceState** indicates whether the source of the data is accepting requests
- **AcceptingRequests** indicates whether the immediate upstream provider (the provider to which the consumer is directly connected) can accept new requests and/or process reissue requests on already open streams.

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

SERVICESTATE	ACCEPTING REQUESTS	MEANING
Up(**1**)	Yes (**1**)	New requests and resissue can be successfully processed.
Up (**1**)	No (**0**)	Although the source 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 accepoting new requests.

Table 29: ServiceState and AcceptingRequests

4.4.3 Service and Group Status Values

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

4.4.3.1 Service Status

The Directory's **ServiceState.Status** Element can be used by providers 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 (Section 2.3) message. EMA consumer implementation normally fans out state from the Status Element to all items associated with the service. When EMA does this, it will not forward this Element to the application. Instead, the application receives a `StatusMsg` for each item from the service. The other elements from the ServiceState FilterEntry will still be sent to the application.

4.4.3.2 Group Status

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

4.4.4 Removing a Services

Provider is able to remove service from a Directory by sending the `MapEntry.Delete` Action for the MapEntry of the service. A consumer should place all open items associated with this service in the `OmmState.ClosedRecover` state. All services associated with a Source Directory stream are removed if:

- The provider sends a message with setting `ClearCache` to **true** on response message for the Directory stream.
- The provider sends a state of `OmmState.Closed` or `OmmState.ClosedRecover` on Source Directory stream.
- The connection between provider and consumer is closed or lost.

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

4.4.5 Service IDs

Most RDM 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.

4.4.6 Automatic Request from EMA Consumer

EMA internal consumer implementation will always automatically request a Directory with the `Filter` set to SERVICE_INFO_FILTER | SERVICE_STATE_FILTER | SERVICE_GROUP_FILTER. This ensures that EMA 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, EMA 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.

Chapter 5 Dictionary Domain

5.1 Description

You can leverage many features within OMM to optimize bandwidth use such as 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 in a file, or available for request over the network from an upstream provider.

There are currently two types of dictionaries with defined 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 then 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 FieldList Contents with Field and Enumerated Types Dictionaries

By itself, a **FieldEntry** contains only the `FieldId` and associated encoded value in **Data**. Because EMA internally stores pre-decoded data, an application can easily decode a FieldEntry (without crossreferencing the `FieldId` to the correct Field Dictionary to determine its type).

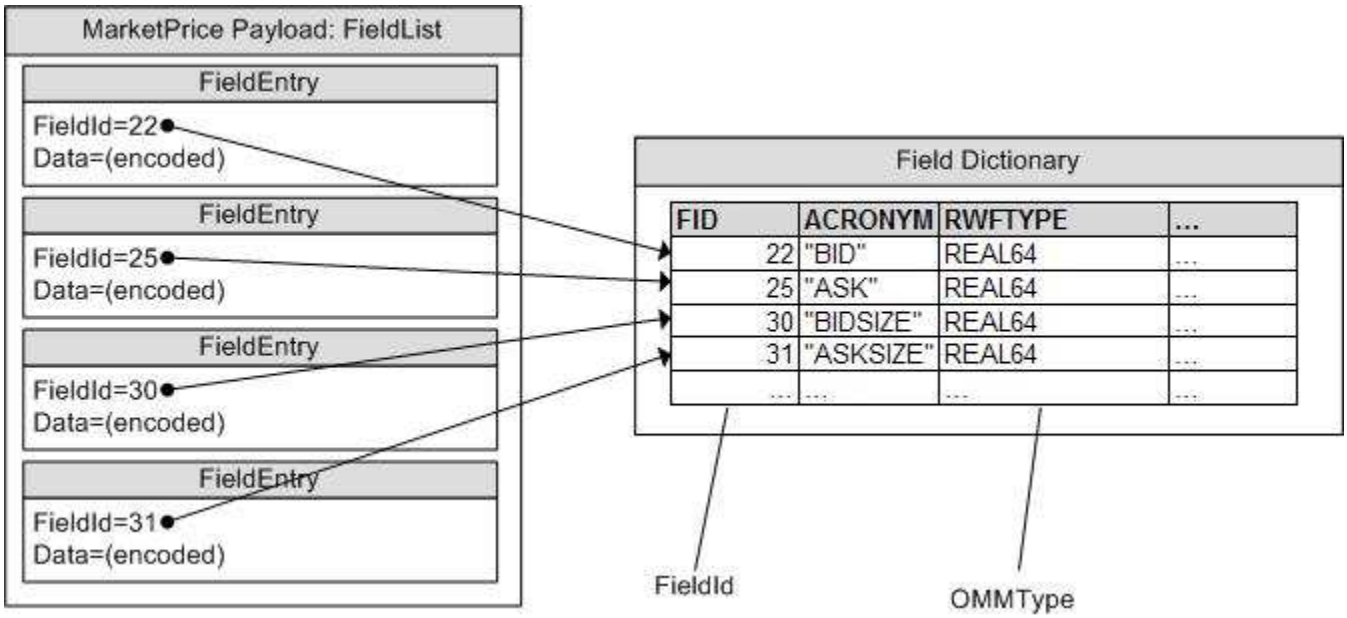

Figure 8: `FieldList` Referencing Field Dictionary

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

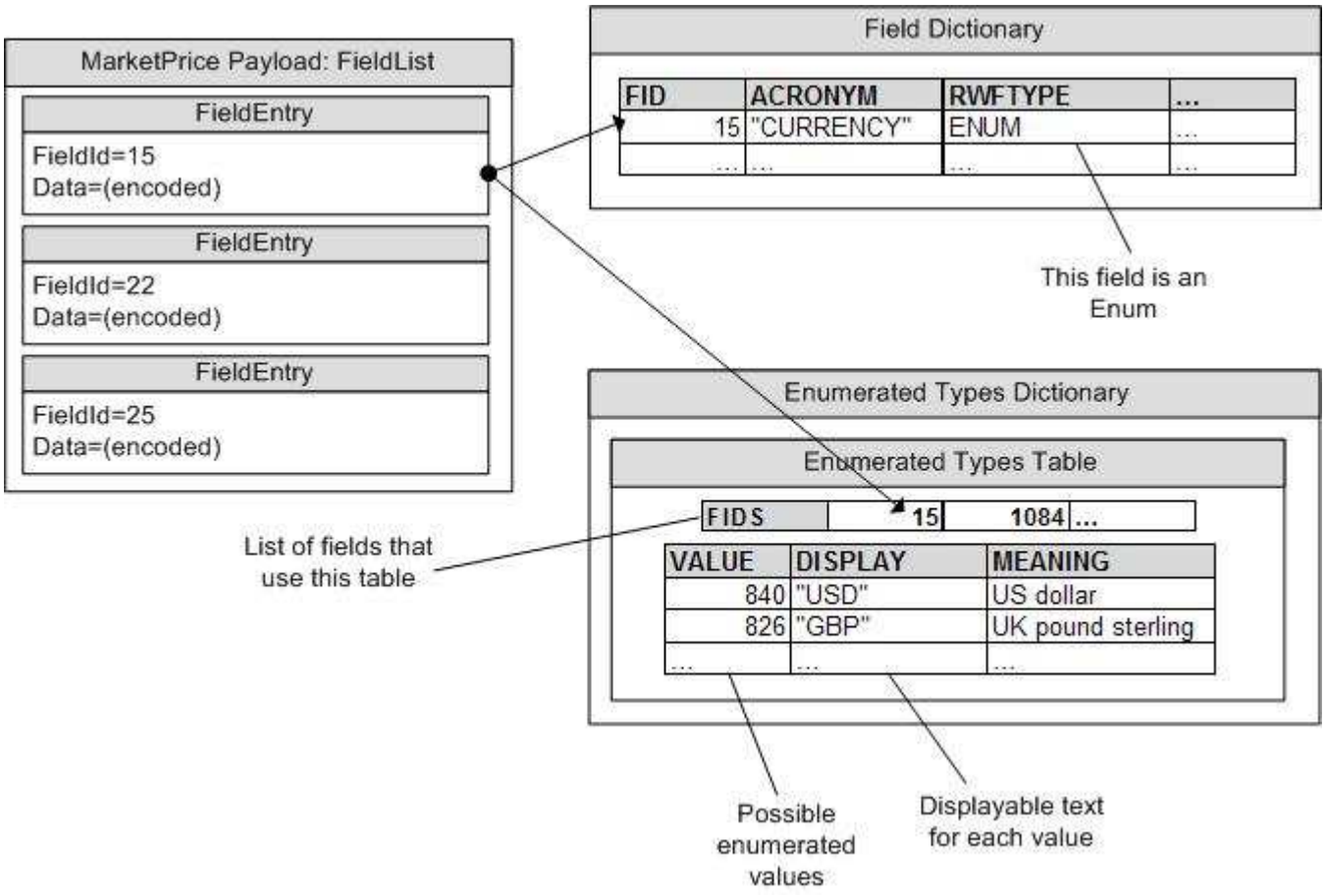

Figure 9: `FieldEntry` Referencing Enumerated Types Table

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

Note: Generic message(s) are not supported for the ***DICTIONARY*** Reuters Domain Model.

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, Thomson Reuters recommends that the consumer make a "streaming request" (setting `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.
`Priority`	Optional.
`Filter`	**Required**. Represents the desired verbosity of the dictionary. The consumer should set the `Filter` according to how much information is needed. • **DICTIONARY_INFO=0x0**: Version information only • **DICTIONARY_NORMAL=0x7**: Provides all information needed for decoding Optional: • **DICTIONARY_MINIMAL=0x3**: Provides information needed for caching • **DICTIONARY_VERBOSE=0xF**: Provides all information(including comments) For details, refer to Section 5.4.1.
`Name`	**Required**. Populate `Name` with the name of the desired dictionary as seen in the Source Directory response (refer to Section 4.3.1.1).
`ServiceName`	**Required**. This will be the service name of a service which listed the Dictionary's Name in the "DictionariesProvided" element of its SeviceInfo **FilterEntry**. **Note:** The application should set either the `ServiceName` or `ServiceId` of the service, but not both.

COMPONENT	DESCRIPTION / VALUE
ServiceId	**Required**. This will be the serviceID of a service which listed the Dictionary's Name in the "DictionariesProvided" element of its SeviceInfo **FilterEntry**. **Note:** The application should set either the ServiceName or ServiceId of the service, but not both.

Table 30: Dictionary Request Message

5.3.2 Dictionary Refresh Message

A Dictionary refresh message is encoded using RefreshMsg and sent by OMM Interactive and non-interactive provider applications and provides the consumer with the content of the requested dictionary.

Since the Referesh messages for Dictionary data might be in multiple Response messages, the provider should only send Name and ServiceName in the first Refresh message. If **MsgKeyInUpdates** is set to **true**, Name and ServiceName must be provided for all Refresh and Status messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_DICTIONARY = 5**
State	**Required**. Contains stream and data state information for the dictionary stream.
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
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
PermissionData	Optional. Used if provided dictionary requires permissioning.

COMPONENT	DESCRIPTION / VALUE
Filter	**Required**. Represents the desired verbosity of the dictionary. • **DICTIONARY_INFO=0x0**: Version information only • **DICTIONARY_NORMAL=0x7**: Provides all information needed for decoding Optional: • **DICTIONARY_MINIMAL=0x3**: Provides information needed for caching • **DICTIONARY_VERBOSE=0xF**: Provides all information(including comments) For details, refer to Section 5.4.1.
Name	**Required**. Populate Name with the name of the desired dictionary as seen in the Source Directory response (refer to Section 4.3.1).
ServiceName	**Required**. This will be the service name of a service which listed the Dictionary's Name in the "DictionariesProvided" element of its SeviceInfo FilterEntry. **Note:** The application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This will be the serviceID of a service which listed the Dictionary's Name in the "DictionariesProvided" element of its SeviceInfo FilterEntry. **Note:** The application should set either the ServiceName or ServiceId of the service, but not both.
Payload	**Required**. The payload structure varies depending on the dictionary's type. However, payload is typically a Series containing an ElementList, while the series summaryData indicates the specific dictionary type and version. The payload could also be XML.

Table 31: Dictionary Refresh Message

5.3.3 Dictionary Status Message

A Dictionary status message is encoded using `StatusMsg` and sent by OMM 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.
`Indications`	Optional: • `DoNotCache` **true**: indicates do not cache • `ClearCache` **true**: indicates clear cache • `Complete` **true**: indicate refresh complete
`PermissionData`	Optional. Used if provided dictionary requires permissioning.

Table 32: Dictionary Status Message

5.4 Data

5.4.1 Filter

While Dictionary's Filters values correlate to a bitmap, only the combinations in Table 33 are supported. 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 33: Dictionary's Filter

5.4.2 Refresh Message Summary Data

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. The SummaryData is extensible and can include other elements.

NAME	TYPE	REQUIRED	DEFAULT	RANGE / EXAMPLE	DESCRIPTION
Version	ASCII	Yes	n/a	e.g. "1.0.1"	Version of the dictionary. Refer to Section 5.8.2. **Note:** The Enumerated Types dictionaries populate the Version element using information present in the DT_Version tag.
Type	UInt	Yes	n/a	**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** Reserved: 0-127 Extensible: 128-255	This can be used to determine what a dictionary is used for and if the dictionary is needed. For example display applications may need style sheets, display templates, and enumeration tables, but caching applications only need field definitions.
DictionaryId	Int	No	0	Unspecified = 0 Reserved: 0 to 16383 Extensible: -1 to -16383 (A **DictionaryId** of **1** corresponds to the RDMFieldDictionary.)	EMA can use **DictionaryId** in field lists and series to associate fields with field definitions or enumerations. Refer to Section 5.4.4.
RT_Version	ASCII	None	No	e.g. "1.0.1"	Optional; sent only with the enumerated type dictionary. This holds information about which field dictionary should be used with this enumerated type dictionary.
DT_Version	ASCII	None	No	e.g. "1.0.1"	Optional; sent only with the enumerated type dictionary. This holds information about the display template's version.

Table 34: Dictionary SummaryData

5.4.3 Response Message Payload

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

Some DictionaryTypes also have external file representations of their data. See sections 5.5.2 and 5.6.2 for details about the data of each DictionaryType.

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 response `Payload.SummaryData` is used to associate 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. **DictionaryId** of **0** means the **DictionaryId** is unspecified, so the Dictionary is not used for parsing or interpreting or displaying any FieldLists. For example, a "TimeZone" Reference Dictionary may include table information about every world time zone. Since this information is not needed to parse FieldLists, there is no reason to assign a DictionaryId to the "TimeZone" Dictionary. So the value of its **DictionaryId** is **0** (i.e. unspecified).

DictionaryIds are globally scoped can have the range of -16383 to 16383. **DictionaryId**s 0 through 16383 are reserved for Reuters use. 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, then it can specify a dictionary switch using 0 for the Field ID. See the *EMA 3.0.4 C++ Edition Developers Guide* for details.

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 contain an `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.

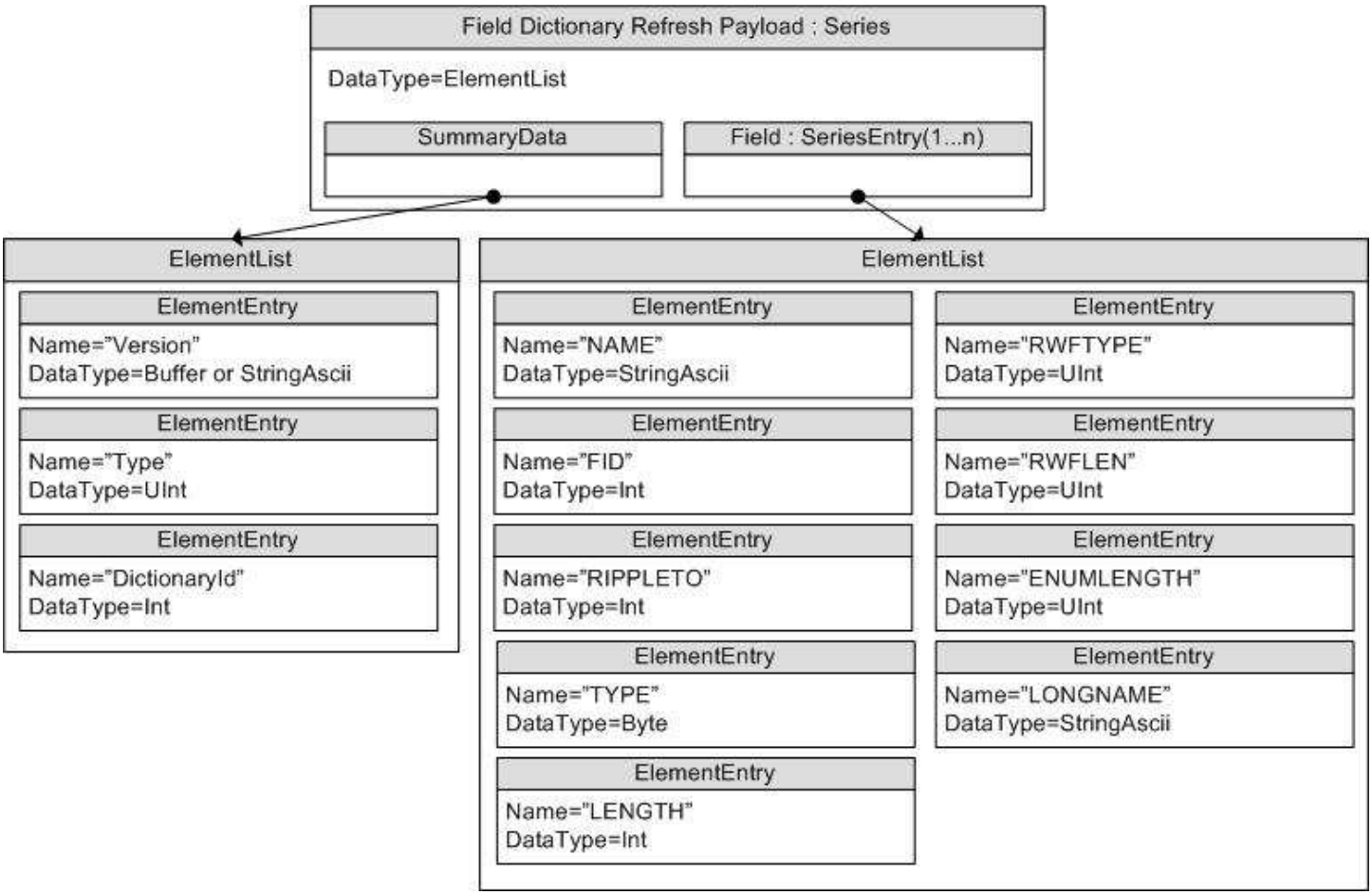

Figure 10: Field Dictionary Payload

NAME	TYPE	DEFAULT	LEAST VERBOSITY	RANGE / EXAMPLE	DESCRIPTION
NAME	StringASCII	None	MINIMAL	e.g. “PROD_PERM”	The field’s short name or acronym.
FID	Int	None	MINIMAL	-32768 to 32767	The field’s `FieldId`.
RIPPLETO	Int	None	MINIMAL	-32768 to 32767.	If the field ripples, this is the `FieldId` of the field it ripples to. A value of 0 sets the field to not ripple. For a description of rippling, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.
TYPE	Int	None	MINIMAL	e.g. INTEGER	The data type of the field for the Marketfeed format.
LENGTH	UInt	None	MINIMAL	0-65535	The maximum string length of the field for the Marketfeed format.
RWFTYPE	UInt	None	MINIMAL	e.g. Int	The `DataType` of the field.
RWFLEN	UInt	None	MINIMAL	0-65535	The maximum length needed to cache the encoded value. This is only a suggestion and is not enforced. A length of **0** implies that it is the maximum possible size for that type.
ENUMLENGTH	UInt	None	NORMAL	0-65535	Used for fields of type **Enumeration**. This is the length of the **DISPLAY** element in its Enumerated Types table (see section 5.6.1).
LONGNAME	ASCII	None	NORMAL	e.g. “PERMISSION”	The “Long Name” or “DDE Acronym” of the field.

Table 35: Field Dictionary `ElementEntry`

5.5.2 Field Dictionary File Format

The **RDMFieldDictionary** file format is a plain text table. Rows are separated by lines and columns are separated by whitespace (excepting quoted strings as shown above). Lines beginning with an exclamation point (!) are comments and are ignored.

Each row represents one field, and each column a datum about that field.

```
!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 11: 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 12: Field Dictionary Tagged Attributes Sample

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

TAG ATTRIBUTE	DESCRIPTION
Filename	The original name of the file as created by Thomson Reuters. 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 Table 34: Dictionary SummaryData. **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 36: Field Dictionary File Tag Information

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

COLUMN NAME IN FILE	RWF 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 RWF type associated with the field (and corresponding to `DataType` values).
RWF LEN	RWFLEN	A caching length hint associated with this field.

Table 37: Field Dictionary File Column Names and ElementEntry Names

5.5.2.1 RWF TYPE Keywords

The following keywords are supported for the RWF TYPE:

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

Table 38: Field Dictionary Type Keywords

5.5.2.2 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**.

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

FIELD TYPE	LENGTH	RWF TYPE	RWF LEN	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, 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 $0\text{-}2^{40}\text{-}1$
INTEGER	15	UINT	8	unsigned int $0\text{-}2^{64}\text{-}1$
INTEGER	15	UINT	4	unsigned int $0\text{-}2^{32}\text{-}1$

Table 39: Marketfeed to RWF Mappings in RDMFieldDictionary

There are a couple of recommendations for custom FIDs:

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 40: Recommended Marketfeed/RWF Field Mappings

These RWF types and values help ensure that data is not truncated when converted from Marketfeed to RWF. If converting RWF to Marketfeed, the OMM Provider application should ensure that the RWF 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**.

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 of the `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.

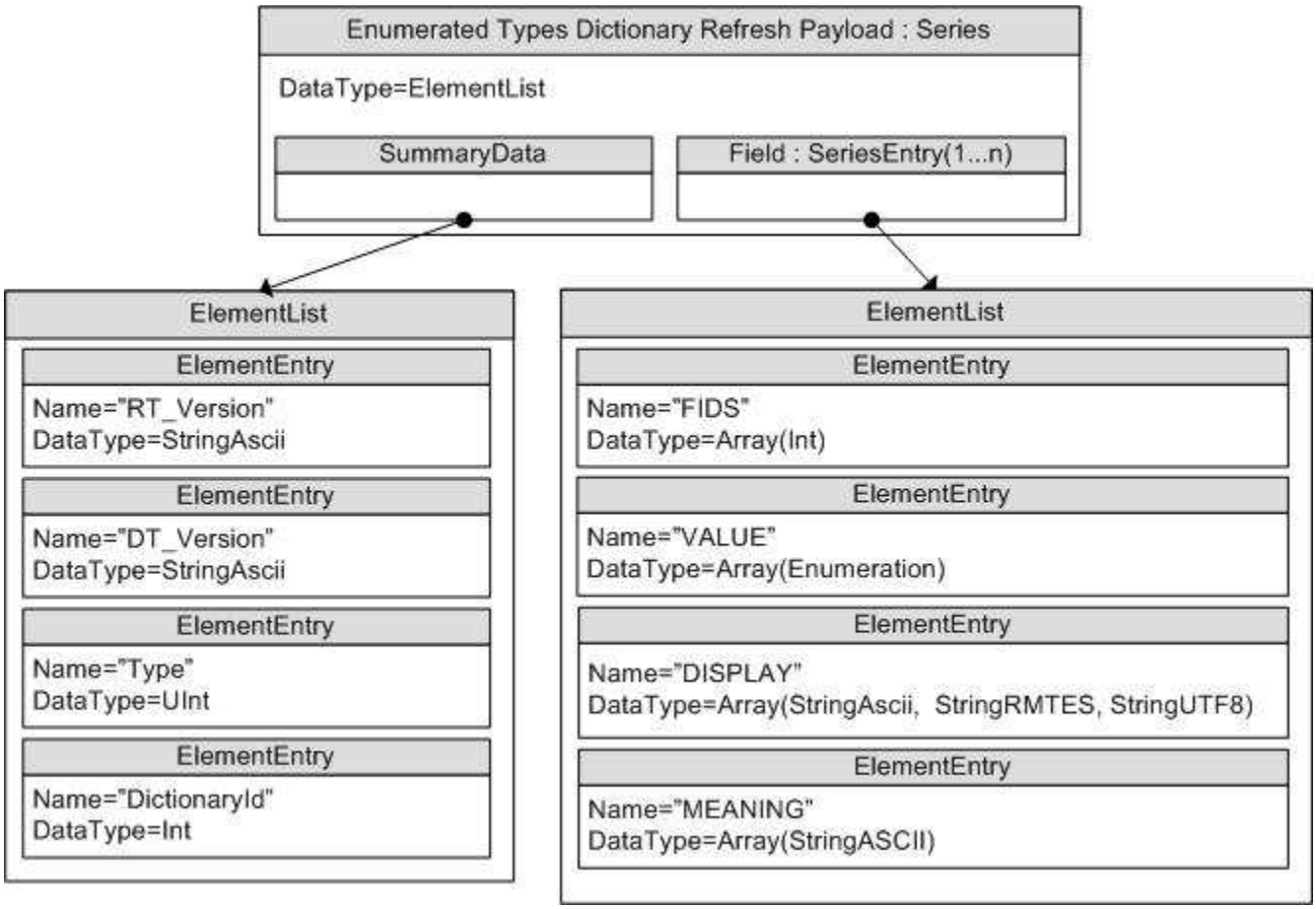

Figure 13: Enumerated Types Dictionary Refresh Message Payload

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 **Enumeration** 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. `FilterEntry` 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 StringASCII	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 41: ElementEntries 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 shown above). Lines beginning with an exclamation point (`!`) are comments and are ignored.

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

1. The list of `FieldId` values corresponding to all fields that use the table
2. The table of enumerated values and their respective display data.

```
! 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
```

Figure 14: Enumerated Types Dictionary File Format Sample

Several tagged attributes are available at the beginning of the file. These attributes provide versioning 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 shown below.

```
!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-May-2015
```

Figure 15: Enumerated Types Dictionary Tagged Attribute Sample

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

TAG ATTRIBUTE	DESCRIPTION
Filename	The original name of the file as created by Thomson Reuters. 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.
Desc	A Description of the dictionary. **Desc** is not used when encoding the domain representation of the field dictionary.
Type	The dictionary type associated with this dictionary. For a field dictionary, this should be **DICTIONARY_ENUM _TABLES=2**. Other types are defined in Table 34: Dictionary SummaryData. **Type** is 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.
DT_Version	The version of the display template version. **DT_Version** is used when encoding the domain representation of the field dictionary. For device compatability purposes, this value is sent as both Version and DT_Version .
Date	Includes information regarding the dictionary release date. **Date** is not used when encoding the domain representation of the field dictionary.

Table 42: Enumerated Type Dictionary File Tag Information

5.6.2.1 Reference Fields Section

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

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

Table 43: RWF EnumType Dictionary File Format Reference Fields

5.6.2.2 Values Table Section

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

NAME	RWF 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 44: RWF 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 Other Dictionary Types

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

None of these dictionary types is currently used.

5.7.1 DisplayTemplate DictionaryType

A DisplayTemplate Dictionary contains specifications that describe how and where to display Fields on a screen.

5.7.2 DataDefinition DictionaryType

A DataDefinition Dictionary contains specifications for ElementListDefs and FieldListDefs that can be used for decoding FieldLists and ElementLists that have been optimized with SetDefinitions.

5.7.3 StyleSheet DictionaryType

A StyleSheet Dictionary contains an XSLT or CSS style sheet.

5.7.4 Reference Dictionary Type

A table of reference information provided as a Series. This information is not used for parsing, interpreting, caching, or displaying data.

5.8 Special Semantics

5.8.1 DictionariesProvided and DictionariesUsed

The Directory's DirectoryInfo FilterEntry (Table 22) includes two Elements that are related to Dictionaries: "DictionariesProvided" and "DictionariesUsed". Both elements contain an `OmmArray` 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.
3. Parse the "DictionariesProvided" from every service in the Directory.
4. Make a streaming request for each Dictionary Name listed in "DictionariesUsed" from the service that lists the DictionaryName in "DictionariesProvided". The DataMask of each request (Section 2.3) message should be set to DICTIONARY_INFO. By only requesting the DICTIONARY_INFO, the application does not incur the extra bandwidth overhead of dictionaries which it does not need.
5. Parse the `Payload.SummaryData` for each Dictionary response message received.
6. If the response `Payload.DataType` is `Series` (of `ElementList`) and DictionaryType `Element` in the `Payload.SummaryData` is not one that the application needs, then unregister the request.
7. If the DictionaryType is needed, then the application should reissue the request with a different DataMask that includes the data (e.g. **DICTIONARY_NORMAL**).
8. If the application previously downloaded and saved the Dictionary, the Version Element can be used to see if the application has the latest version.
    - If the application does not have the latest version, the application should reissue the request for the Dictionary with a different DataMask.
    - If the application has the latest version, it can leave the first request open so it can monitor when the dictionary is changed.

5.8.2 Version Check

Dictionary version checking can be performed by the client after a refresh (Section 2.3) response message of a Dictionary is received. The version information is available in the `SummaryData` of the payload. An application that has a previously cached dictionary can request only the **DICTIONARY_INFO**. That way, the application does not incur the extra bandwidth overhead from downloading a dictionary which it already has.

Version will be an ASCII or Buffer. It is a major number, minor number, separated by a period (e.g. “1.2”). A third, micro number is optional (e.g. “1.2.1”). The major number changes when an incompatible change is made, such as changing a field’s type or length. The minor number changes when a compatible change occurs, such as adding a field, changing a field’s name, or removing a field (but ensuring it is not reused). The micro number would change for comment changes.

If two services provide the same Dictionary with different minor versions, the latest version should be used.

5.8.3 Streaming Dictionary

Dictionary request can be streaming. Dictionary providers are not allowed to send refresh and update data to consumers. Instead the provider can advertise a minor Dictionary change by sending a status (Section 2.3) response message with a DataState of Suspect. It is the consumer’s responsibility to reissue the dictionary request.

If the consumer does not reissue the request for the new dictionary it will still be able to process other data streams. However, the application will still be responsible for realizing when unknown fields are encountered. The application must skip those fields.

If the dictionary has a major version change, the provider must either disconnect its clients’ network connections or it must send an `OmmState.ClosedRecover` status message for each item or group that uses the data.

5.9 Specific Usage: Enterprise Platform

The Enterprise Platform currently only supports a single DictionaryId’s family. If the provider doesn’t specify it then it is interpreted to be 1.

Chapter 6 MarketPrice Domain

6.1 Description

The MarketPrice 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 the information related to that item (i.e., net change, bid, ask, volume, high, low, or last price).

6.2 Usage

Note: Generic Message(s) are not supported in the **MMT_MARKET_PRICE** Reuters Domain Model.

6.2.1 MarketPrice Request Message

A MarketPrice request message is encoded using `ReqMsg` and sent by OMM consumer applications. The request specifies the name and attributes of an item in which the consumer is interested.

If a consumer wishes to receive updates, it can make a “streaming” request by setting `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).

A consumer can pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_PRICE = 6**
`Interactions`	**Required**. • `InitialImage` **true**: indicates initial image is required • `InterestAfterRefresh` **true**: indicates streaming request is required • `Pause` **true**: indicates 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 quality of service at which the consumer want the stream serviced.
`Priority`	Optional. Indicates class and count associated with stream priority.

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. The name of requested item. **Note:** Not used for Batch Item request.
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The consumer application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The consumer application should set either the ServiceId or ServiceName of the service, but not both.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. See Chapter 12 for more detailed information about View.

Table 45: MarketPrice Request Message

6.2.2 MarketPrice Refresh Message

A MarketPrice Refresh Message is encoded using RefreshMsg and sent by OMM Provider and OMM non-interactive provider applications. This message sends all currently available information about the item to the consumer.

FieldList in the payload should include all the 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 the 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 MarketPrice domain must be atomic. The MarketPrice 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
MessageModelType	**Required**. **MMT_MARKET_PRICE = 6**
State	Required. Includes the state of the stream and data.
Solicited	**Required**. • **true**: indicate refresh solicited • **false**: indicate refresh unsolicited
Indications	**Required**. • Complete **true**: indicate refresh complete Optional: • DoNotCache **true**: indicate do not cache this refresh message • ClearCache **true**: indicate clear cache
QoS	Optional. If sent, specifies the quality of service which the stream is provided.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
ItemGroup	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies the permission information associated with content of this stream.
NameType	Optional. This should match name type specified in the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. This should match the requested name.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. Specifies the name of the service from which the cosumer wishes to request 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 from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceId or ServiceName of the service, but not both.
Payload	**Required**. This should consist of a FieldList containing all fields associated with the item.

Table 46: MarketPrice Refresh Message

6.2.3 MarketPrice Update Message

A MarketPrice Update Message is encoded using `UpdateMsg` and sent by OMM Provider and OMM non-interactive provider applications. It 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 messages.

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	**Required**. `DoNotRipple` is set to **true**. If `UpdateTypeNum` is set to be **INSTRUMENT_UPDATE_CORRECTION=7** or **INSTRUMENT_UPDATE_VERIFY=11** Optional: • `DoNotCache` **true**: Indicates the application should not cache this update message. • `ClearCache` **true**: Indicates the application should clear the cache. • `DoNotConflate` **true**: Indicates the application should not conflate updates.
`QoS`	Optional.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`ConflatedCount`	Optional. If a provider is sending a conflated update, this informs the consumer of how many updates are in the conflation. The consumer indicates interest in this information by setting the `ReqMsg.ConflatedInUpdates` to **true** in the request.

COMPONENT	DESCRIPTION / VALUE
ConflatedTime	Optional. If a provider is sending a conflated update, this notifies the consumer of the times interval(in milliseconds) over which data is conflated. The consumer indicates interest in this information by setting ReqMsg.ConflatedInUpdates to **true** in the request.
ItemGroup	Optional
PermissionData	Optional. Permissioning information associated with only the contents of this update.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
Payload	**Required**. This should consist of a FieldList with any changed data.

Table 47: MarketPrice Update Message

6.2.4 MarketPrice Status Message

A MarketPrice status message is encoded using `StatusMsg` and sent by OMM 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`, `ServiceName` in the first Refresh response message. However if **MsgKeyInUpdates** is set to **true**, then `Name` and `ServiceName` must be provided for **every** Status response messages.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_PRICE = 6**
`State`	Optional. Current state information associated with the data and stream.
`Indications`	Optional: • **ClearCache** **true**: indicates clear cache
`QoS`	Optional.
`SeqNum`	Optional.
`ConflatedCount`	Optional.
`ConflatedTime`	Optional.
`ItemGroup`	Optional. The provider can use this component to change the item's `ItemGroup`
`PermissionData`	Optional. If present, this is the new permission information associated with all the contents on the stream.
`NameType`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). Specifies the name of the item being provided.
`ServiceName`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). Specifies the name of the service providing the data. **Note:** The provider application should set either the `ServiceName` or `ServiceId` of the service, but not both.
`ServiceId`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). Specifies the ID of the service providing the data. **Note:** The provider application should set either the `ServiceName` or `ServiceId` of the service, but not both.

Table 48: MarketPrice Status Message

6.2.5 MarketPrice Post Message

If support is specified by the provider, consumer applications can post the MarketPrice data. For more details on posting, see the *EMA 3.0.4 C++ Edition Developers Guide*.

6.3 Data: Response Message Payload

Market Price data is conveyed as a `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 *EMA 3.0.4 C++ Edition Developers Guide*.

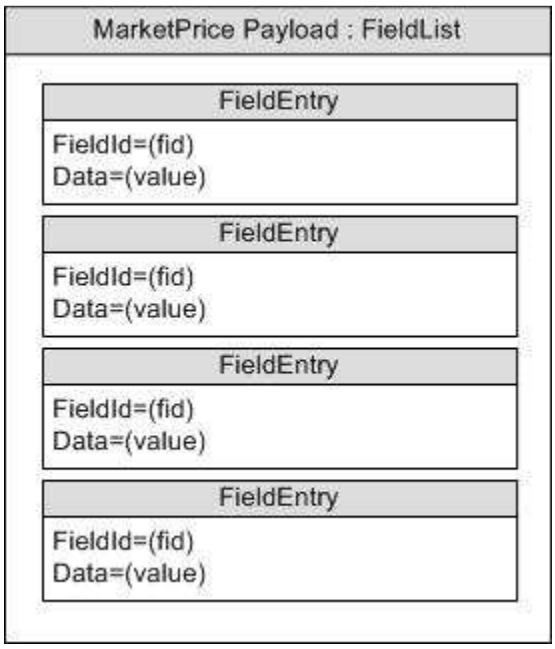

Figure 16: MarketPrice Response Message Payload

6.4 Special Semantics

6.4.1 Snapshots

MarketPrice is one of a few message model types that support a true snapshot. If a nonstreaming request is made, then the `UpdateMsg` will not be sent. Status messages could be received before the single Refresh response (Section 2.3) message 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 the FieldList are defined as ripple Field. 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 EMA does NOT ripple fields on behalf of the Consumer application. The OMM `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 that is structured like IDN records. This includes:

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

Chapter 7 MarketByOrder Domain

7.1 Description

The MarketByOrder 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).

7.2 Usage

Note: Generic Message(s) are not supported for **MMT_MARKET_BY_ORDER** Reuters Domain Model.

7.2.1 MarketByOrder Request Message

A MarketByOrder request message is encoded using `ReqMsg` and sent by OMM consumer applications. The request specifies the name of the item in which a consumer is interested.

If a consumer wishes to receive updates, it may make a "streaming" request by setting the `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).

A consumer can send pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_BY_ORDER = 7**
`Interactions`	**Required**. • `InitialImage` **true**: indicates initial image is required. • `InterestAfterRefresh` **true**: indicates streaming request is required. • `Pause` **true**: indicates pause is required.
`Indications`	Optional. • `ConflatedInUpdates` **true** : indicates conflated in updates
`QoS`	Optional. Indicates the quality of service at which the consumer want the stream serviced at.
`Priority`	Optional. Indicates class and count associated with stream priority
`NameType`	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.

COMPONENT	DESCRIPTION / VALUE
Name	**Required**. The name of requested item. **Note:** Not used for Batch Item request.
ServiceName	**Required**. **Note:** The consumer application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. **Note:** The consumer application should set either the ServiceId or ServiceName of the service, but not both.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. See Chapter 12 for more detailed information about View.

Table 49: MarketByOrder Request Message

7.2.2 MarketByOrder Refresh Message

A MarketByOrder refresh message is encoded using `RefreshMsg` and sent by OMM interactive provider and OMM non-interactive provider applications. A MarketByOrder 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 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_BY_ORDER = 7**
`State`	Required. Includes the state of the stream and data.
`Solicited`	**Required.** • **true**: indicates refresh solicited • **false**: indicates refresh unsolicited
`Indications`	Optional: • `DoNotCache` **true**: indicate do not cache this refresh message • `ClearCache` **true**: indicate clear cache • `Complete` **true**: indicate refresh complete
`QoS`	Optional. If sent, specifies the quality of service which the stream is provided.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`PartNum`	Optional. A user-specified, item-level number that indicates the part number of a multi-part refresh.
`ItemGroup`	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
`PermissionData`	Optional. Specifies the permission information associated with content of this stream.
`NameType`	Optional. This should match name type specified in the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	**Required**. This should match the requested name.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. Spceifies the name of the service from which the cosumer wishes to request 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 from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceId or ServiceName of the service, but not both.
Payload	**Required**. An orderbook is represented by a Map, where each entry (MapEntry) contains information (FieldList) that corresponds to an order.

Table 50: MarketByOrder Refresh Message

7.2.3 MarketByOrder Update Message

A MarketByOrder update message is encoded using `UpdateMsg` and sent by OMM interactive provider and OMM 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 updates. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED=0** • **INSTRUMENT_UPDATE_QUOTE=1**
`Indications`	Optional: • `DoNotCache` **true**: Indicates the application should not cache this update message. • `ClearCache` **true**: Indicates the application should clear its cache. • `DoNotConflate` **true**: Indicates 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 with this stream.
`ConflatedCount`	Optional. If a provider is sending a conflated update, this informs the consumer of 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, this notifies the consumer of the times 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.
`NameType`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.

COMPONENT	DESCRIPTION / VALUE
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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.
Payload	**Required**. An orderbook is represented by a Map, where each entry (MapEntry) contains information (FieldList) that corresponds to an order.

Table 51: MarketByOrder Update Message

7.2.4 MarketByOrder Status Message

A ***MarketByOrder status*** message is encoded using `StatusMsg` and sent by OMM 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. Current state information associated with the data and stream.
`Indications`	Optional: • `ClearCache` **true**: indicate clear cache
`SeqNum`	Optional.
`ItemGroup`	Optional. The provider can use this component to change the item's Group (See Section 4.3.1.3).
`PermissionData`	Optional. If present, this is the new permission information associated with all the contents on the stream.
`NameType`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). Specifies the name of the item being provided.
`ServiceName`	Optional (**Required** if **MsgKeyInUpdates** is 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`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). Specifies the ID of the service providing data. **Note:** The application should set either the `ServiceName` or `ServiceId` of the service, but not both.

Table 52: MarketByOrder Status Message

7.2.5 MarketByOrder Post Message

If support is specified by the provider, consumer applications can post MarketByOrder data. For more details on posting, see the *EMA 3.0.4 C++ Edition Developers Guide*.

7.3 Data: 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. Since the fields in each `MapEntry` are the same, bandwidth can be further optimized by DataDefinitions.

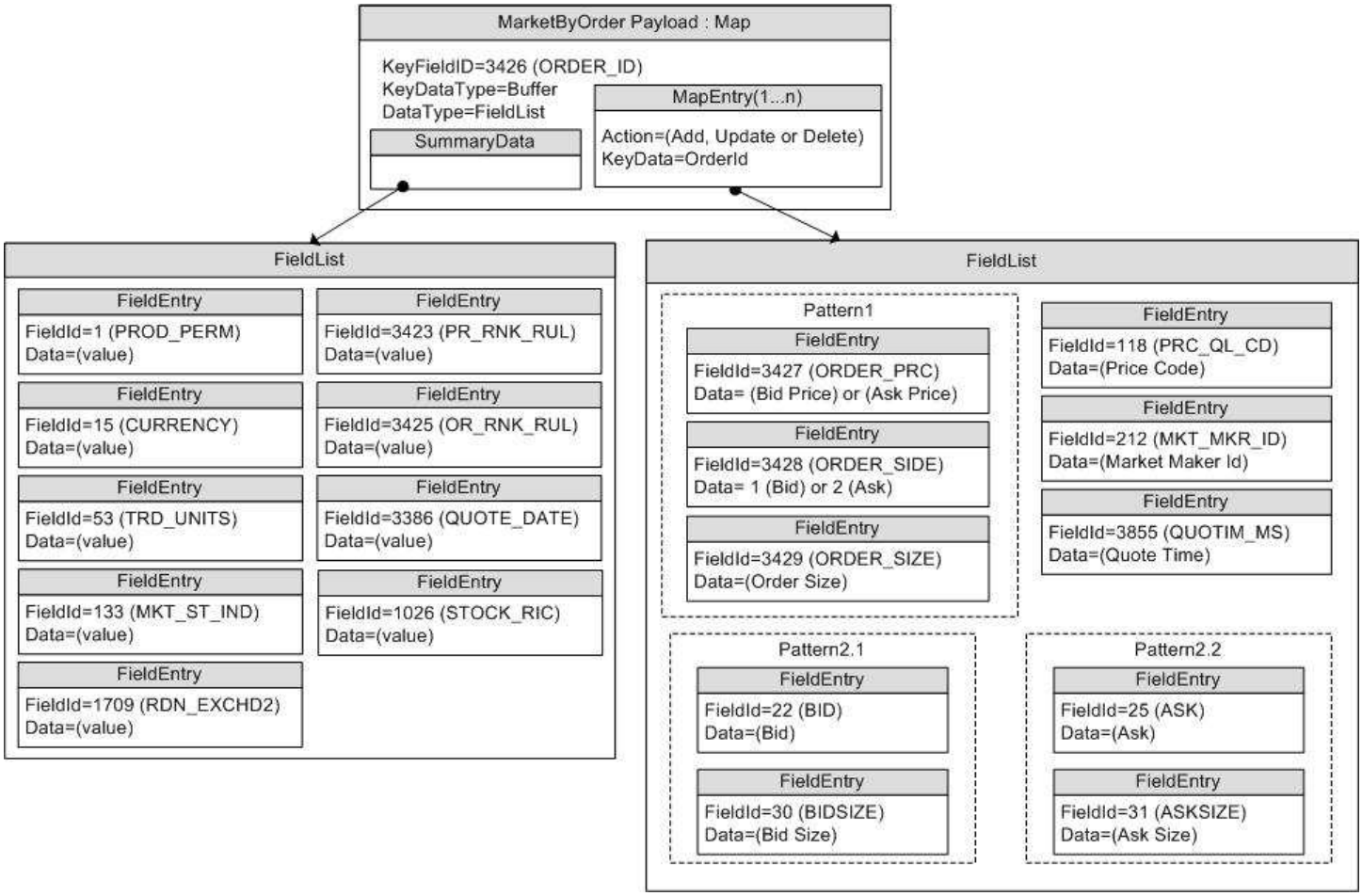

Figure 17: MarketByOrder Response Message Payload

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.

The **Map.SummaryData** only needs to be in the first refresh msg which typically includes:

- Permission information (PROD_PERM)
- Currency of the orders (CURRENCY)
- Trade Units for the precision for which order prices are set (TRD_UNITS)
- Market State (MKT_ST_IND)
- Exchange Identifier 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)

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 & Side (BID, ASK, or ORDER_PRC & 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 SummaryData and a single `MapEntry`. Updates are not sent for any map entry until after thea 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: Enterprise Platform

For the most part, MarketByOrder data from the Enterprise Platform is the same as it is from the original source of the data (e.g. RDF Direct). However, if caching is enabled in an Enterprise Platform component, there are two differences.

- The number of messages packed into each Refresh response message may be different.
- An updated reponse 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.

Chapter 8 MarketByPrice Domain

8.1 Description

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

8.2 Usage

Note: Generic Message(s) are not supported for the **MMT_MARKET_BY_PRICE** Reuters Domain Model.

8.2.1 MarketByPrice Request Message

A MarketByPrice request message is encoded using `ReqMsg` and sent by OMM consumer applications. The request specifies the name of an item in which the consumer is interested.

If a consumer wishes to receive updates, it 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 send pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_BY_PRICE = 8**
`Interactions`	**Required**. • `InitialImage` **true**: indicates initial image is required • `InterestAfterRefresh` **true**: indicates streaming request is required • `Pause` **true**: indicates 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 quality of service at which the consumer want the stream serviced.
`Priority`	Optional. Indicates class and count associated with stream priority.

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. The name of requested item. **Note:** Not used for Batch Item request.
ServiceName	**Required**. **Note:** The application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. **Note:** The application should set either the ServiceId or ServiceName of the service, but not both.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. See Chapter 12 for more detailed information about View.

Table 53: MarketByPrice Request Message

8.2.2 MarketByPrice Refresh Message

A MarketByPrice refresh message is encoded using RefreshMsg and sent by OMM interactive provider and OMM non-interactive provider applications.

A MarketByPrice 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 streaming or non-streaming request.

Note: The provider should only send the Name and ServiceName in the first Refresh response message. However if EMA is configured **MsgKeyInUpdates** with **true** value, then the Name and ServiceName must be provided for **every** Refresh response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
State	**Required**. Includes the state of the stream and data.
Solicited	**Required.** • **true**: Indicates the refresh message is solicited. • **false**: Indicates the refresh message is unsolicited.
Indications	Optional: • DoNotCache **true**: Indicates the application should not cache this refresh message. • ClearCache **true**: Indicates the application should clear its cache. • Complete **true**: Indicates this is the last message in the refresh complete.
QoS	Optional. If sent, specifies the quality of service which the stream is provided.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
PartNum	Optional. A user-specified, item-level number that indicates the part number in a multi-part refresh.
ItemGroup	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
PermissionData	Optional. Specifies the permission information associated with content of this stream.
NameType	Optional. This should match name type specified in the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. This should match the requested name.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The consumer application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The consumer application should set either the ServiceId or ServiceName of the service, but not both.
Payload	**Required**. An orderbook is represented by a Map, where each entry (MapEntry) contains FieldList which has information about a price point.

Table 54: MarketByPrice Refresh Message

8.2.3 MarketByPrice Update Message

A MarketByPrice update message is encoded using `UpdateMsg` and sent by OMM interactive provider and OMM 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 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_PRICE = 8**
`UpdateTypeNum`	**Required**. Indicates the general content of the updates. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED=0** • **INSTRUMENT_UPDATE_QUOTE=1**
`Indications`	Optional: • `DoNotCache` **true**: indicates do not cache this update message • `DoNotConflate` **true**: indicates do not conflate this update message
`QoS`	Optional.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`ConflatedCount`	Optional. If a provider is sending a conflated update, this informs the consumer of 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, this notifies the consumer of the times 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.
`NameType`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.

COMPONENT	DESCRIPTION / VALUE
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
Payload	**Required**. MarketByPrice is represented by a Map, where each entry (MapEntry) contains FieldList which has information about a price point.

Table 55: MarketByPrice Update Message

8.2.4 MarketByPrice Status Message

A MarketByPrice status message is encoded using StatusMsg and sent by OMM 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 if EMA is configured with **MsgKeyInUpdates** set to **true**, then the Name and ServiceName must be provided for **every** Status response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_BY_PRICE = 8**
State	Optional. Current state information associated with the data and stream.
Indications	Optional: • ClearCache **true**: indicates clear cache
QoS	Optional.
ItemGroup	Optional. The provider can use this component to change the item's ItemGroup
PermissionData	Optional. If present, this is the new permission information associated with all the contents on the stream.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.

Table 56: MarketByPrice Status Message

8.2.5 MarketByPrice Post Message

If support is specified by the provider, consumer applications can post MarketByPrice data. For more details on posting, see the *EMA 3.0.4 C++ Edition Developers Guide*.

8.3 Data: 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.

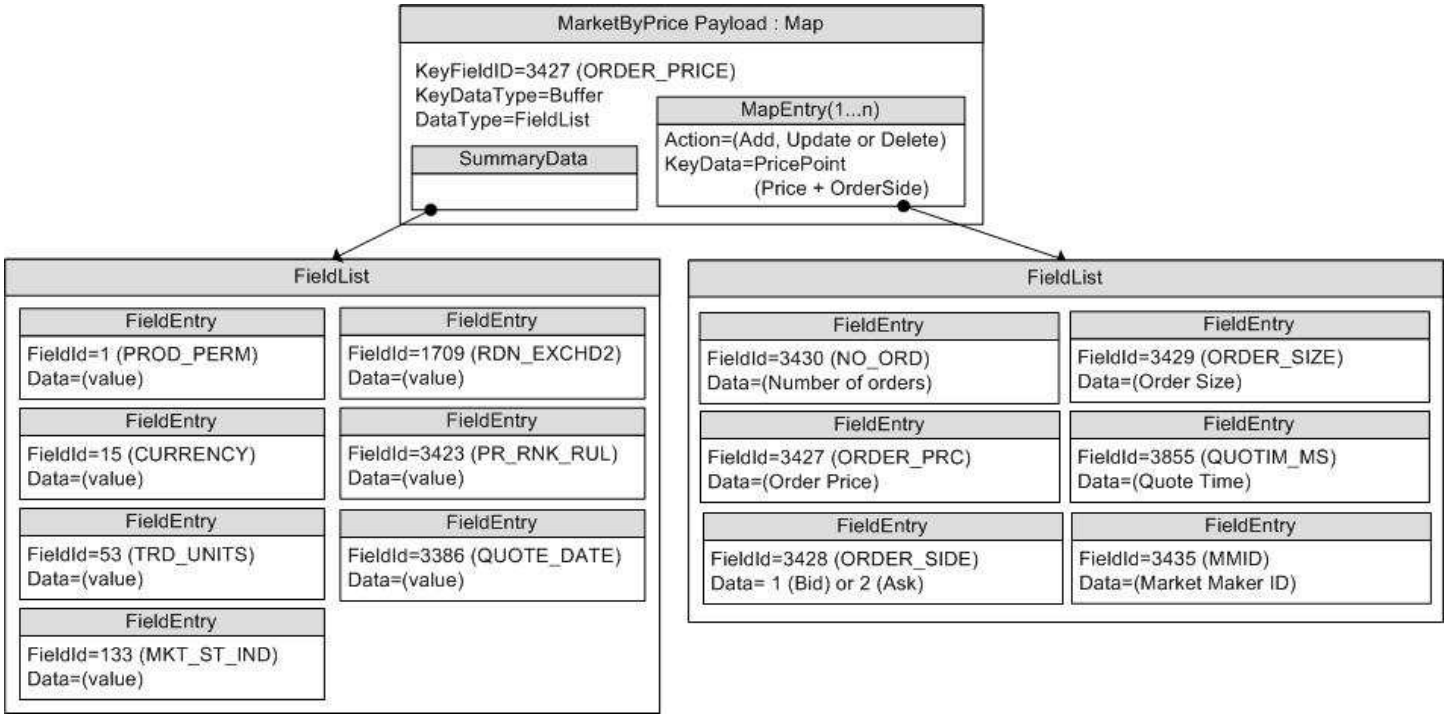

Figure 18: MarketByPrice Response Message Payload

The **Map.SummaryData** only needs to be in the first refresh msg which typically includes:

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

The **MapEntry.Key**'s data is a Buffer that contains the combination of the price and the order side, 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.Data** is a `FieldList` that contains some or all of the following information about the price point:

- Number of aggregated orders (NO_ORD)
- Order Price & Side (BID, ASK, or ORDER_PRC & ORDER_SIDE)
- Order Size (BIDSIZE, ASKSIZE, or ORDER_SIZE)
- Quote Time (QUOTIM_MS)
- Map containing the Market Makers (MMID) and optionally a field list with the positions of each market maker 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: Enterprise Platform

For the most part, MarketByPrice data from the Enterprise Platform is the same as it is from the original source of the data (e.g. RDF Direct). However, if caching is enabled in an Enterprise Platform 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.

Chapter 9 MarketMaker Domain

9.1 Description

The ***MarketMaker*** 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 `FieldList` describing information such as that market maker's bid and ask prices, quote time, and market source.

9.2 Usage

Note: Generic Message(s) are not supported for the **MMT_MARKET_MAKER** Reuters Domain Model.

9.2.1 MarketMaker Request Message

A MarketMaker request message is encoded using `ReqMsg` and sent by OMM consumer applications. The request specifies the name of an item in which the consumer is interested.

If a consumer wishes to receive updates, it may make a "streaming" request by setting the `ReqMsg.InterestAfterRefresh` is set to **true**. If the flag 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).

A consumer can send pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_MAKER = 9**
`Interactions`	**Required**. • `InitialImage` **true**: indicates initial image is required • `InterestAfterRefresh` **true**: indicates streaming request is required • `Pause` **true**: indicates 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 quality of service at which the consumer want the stream serviced.
`Priority`	Optional. Indicates class and count associated with stream priority.

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. The name of requested item. **Note:** Not used for Batch Item request.
ServiceName	**Required**. This should be the Name associated with 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.
ServiceId	**Required**. This should be the ID associated with the service from which the consumer wishes to request the item. **Note:** The consumer application should set either the ServiceId or ServiceName of the service, but not both.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. For more detailed information about View see Chapter 12.

Table 57: MarketMaker Request Message

9.2.2 MarketMaker Refresh Message

A MarketMaker refresh message is encoded using `RefreshMsg` and sent by OMM interactive provider and OMM non-interactive provider applications.

The MarketMaker refresh can be sent in multiple parts. It is possible for update and status messages to be delivered between parts of a refresh message, regardless of streaming or non-streaming request.

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** Refresh response messages.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_MAKER = 9**
`State`	**Required**. Includes the state of the stream and data.
`Solicited`	**Required.** • **true**: Indicates the message was solicited • **false**: Indicates the message is unsolicited
`Indications`	Optional: • `DoNotCache` **true**: Requests the application to not cache • `ClearCache` **true**: Requests that the application clear the cache • `Complete` **true**: Indicates the message is the final one in the refresh.
`QoS`	Optional. If sent, specifies the quality of service which the stream is provided.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`PartNum`	Optional. A user-specified, item-level number that indicates the part number in a multi -part refresh.
`ItemGroup`	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
`PermissionData`	Optional. Specifies the permission information associated with content of this stream.
`NameType`	Optional. This should match name type specified in the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	**Required**. Symbol for MarketMaker Item.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceId or ServiceName of the service, but not both.
Payload	**Required**. A MarketMaker is represented by a Map, where each entry (MapEntry) contains FieldList which has information about a market maker.

Table 58: MarketMaker Refresh Message

9.2.3 MarketMaker Update Message

A MarketMaker update message is encoded using UpdateMsg and sent by OMM interactive provider and OMM 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 only send the Name and ServiceName in the first Refresh response message. However if EMA is configured **MsgKeyInUpdates** with **true** value, then the Name and ServiceName must be provided for **every** Update response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_MARKET_MAKER = 9**
UpdateTypeNum	**Required**. Indicates the general content of the updates. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED=0** • **INSTRUMENT_UPDATE_QUOTE=1**
Indications	Optional: • DoNotCache **true**: indicates do not cache this update message • ClearCache **true**: indicates clear cache • DoNotConflate **true**: indicates do not conflate this update message
QoS	Optional.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
PartNum	Not used.
ConflatedCount	Optional. If a provider is sending a conflated update, this informs the consumer of 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 is sending a conflated update, this notifies the consumer of the times 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. Permissioning information associated with only the contents of this update.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.

COMPONENT	DESCRIPTION / VALUE
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID 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	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
Payload	**Required**. A MarketMaker is represented by a Map, where each entry (MapEntry) contains FieldList which has information about a market maker.

Table 59: MarketMaker Update Message

9.2.4 MarketMaker Status Message

A MarketMaker status message is encoded and sent by OMM 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 if EMA is configured **MsgKeyInUpdates** with **true** value, then the `Name` and `ServiceName` must be provided for **every** Status response messages.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_MARKET_MAKER = 9**
`State`	Optional. Current state information associated with the data and stream.
`Indications`	Optional: • `ClearCache` **true** : indicates clear cache
`QoS`	Optional.
`ItemGroup`	Optional. The provider can use this component to change the item's `ItemGroup`.
`PermissionData`	Optional. If present, this is the new permission information associated with all the contents on the stream.
`NameType`	Optional (**Required** if EMA is configured **MsgKeyInUpdates** with **true** value). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if EMA is configured **MsgKeyInUpdates** with **true** value). This should match the name of the item being provided.
`ServiceName`	Optional (**Required** if EMA is configured **MsgKeyInUpdates** with **true** value). This should be 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`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the `ServiceName` or `ServiceId` of the service, but not both.

Table 60: MarketMaker Status Message

9.2.5 MarketMaker Post Message

If support is specified by the provider, consumer applications can post MarketMaker data. For more details on posting, see the *EMA 3.0.4 C++ Edition Developers Guide*.

9.3 Data: 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. Since the fields in each map entry are the same, bandwidth can be further optimized by DataDefinitions.

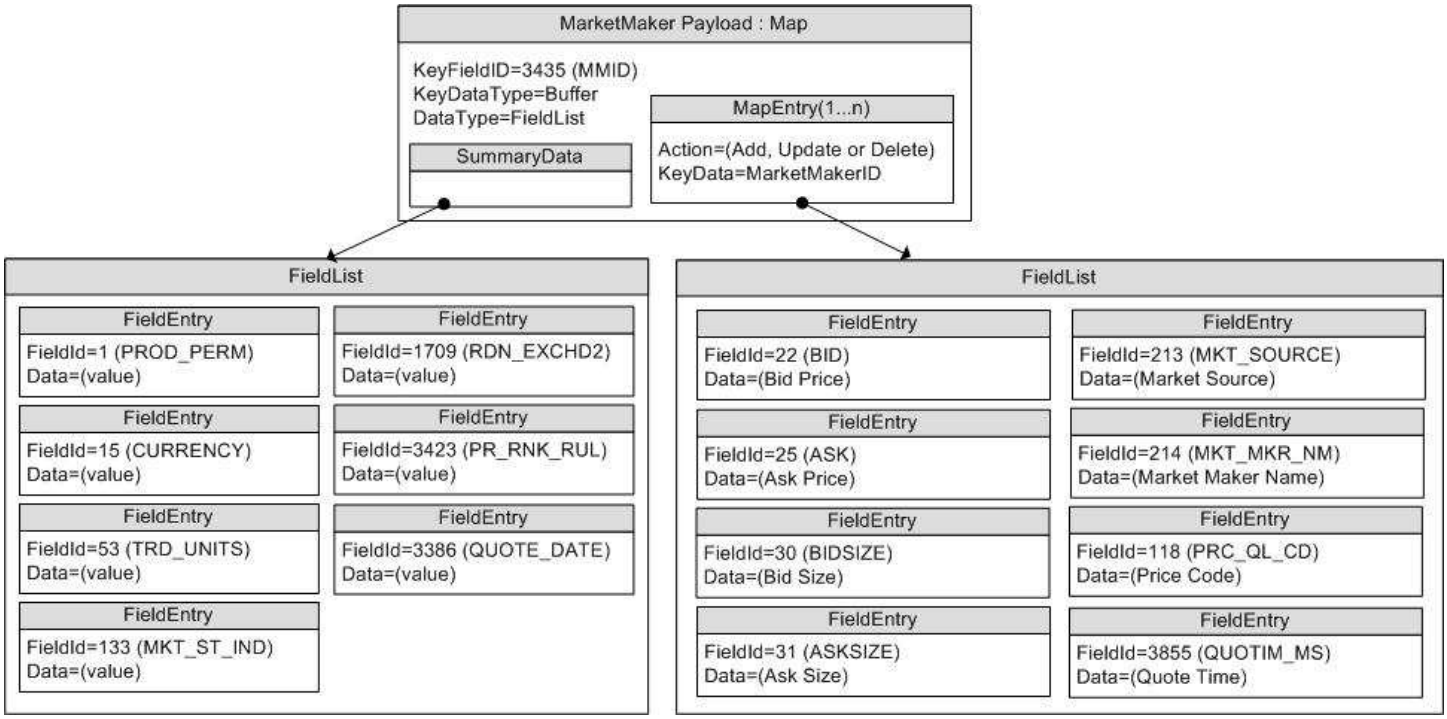

Figure 19: MarketMaker Response Message Payload

The **Map.SummaryData** only needs to be in the first refresh msg which typically includes:

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

The **MapEntry.Key**'s data is a Buffer, AsciiString or RmtesString containing a unique market maker 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`.

The `MapEntry.Data` is a `FieldList` that contains the following information about the top bid and ask order for a market maker:

- 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 & 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 **true** value. 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: Enterprise Platform

For the most part, MarketMaker data from the Enterprise Platform is the same as it is from the original source of the data (e.g. RDF Direct). However, if caching is enabled in an Enterprise Platform component, there will be two differences.

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

ANnupdates response message may be delivered between Refresh response messages, before the `RefreshMsg.Complete` is sent with a **true** value. It is the consumer applications responsibility to apply the indicated changes.

Chapter 10 YieldCurve Domain

10.1 Description

This section defines a yield curve model as is currently supported by the ATS. The ***YieldCurve*** domain shows the relation between the interest rate and the term, or time to maturity, associated with the debt of a borrower. The shape of the yield curve can help to give an idea of future economic activity and interest rates. Information is sent as an `FieldList`, where some `FieldEntry`'s may contain more complex types such as `Vector`, `Array`, or `FieldList`.

10.2 Usage

Note: `GenericMsg`(s) are not supported in the **MMT_YIELD_CURVE** Reuters Domain Model.

10.2.1 YieldCurve Request Message

A YieldCurve request message is encoded using `ReqMsg` and sent by OMM consumer applications. The request specifies the name and attributes of the curve in which the consumer is interested.

If a consumer wishes to receive updates, it may make a "streaming" request by setting the `ReqMsg.InterestAfterRefresh` to **true**. If the flag 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).

A consumer can send pause an item to stop updates (if the provider supports such functionality). For more information, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_YIELD_CURVE = 22**
`Interactions`	**Required**. • `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 quality of service at which the consumer want the stream serviced.
`Priority`	Optional. Indicates class and count associated with stream priority.

COMPONENT	DESCRIPTION / VALUE
NameType	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	**Required**. The name of requested item. **Note:** Not used for Batch Item request.
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The application should set either the service's ServiceId or ServiceName, but not both.
Payload	Optional. When features such as View or Batch are leveraged, the payload can contain information relevant to that feature. See Chapter 12 for more detailed information about View.

Table 61: YieldCurve Request Message

10.2.2 YieldCurve Refresh Message

A YieldCurve Refresh Message is encoded using `RefreshMsg` and sent by OMM Provider and OMM non-interactive provider applications. This message sends all currently available information about the item to the consumer.

`FieldList` in the payload should include all the 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 the 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: 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 Refresh response messages.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_YIELD_CURVE = 22**
`State`	**Required**. Includes the state of the stream and data.
`Solicited`	**Required.** • **true**: Indicates a refresh solicited • **false**: Indicates a refresh unsolicited
`Indications`	Optional: • `DoNotCache` **true**: Indicates do not cache for this refresh message • `ClearCache` **true**: Requests the application to clear the cache. • `Complete` **true**: Indicates that the message completes the refresh.
`QoS`	Optional. If sent, specifies the quality of service which the stream is provided.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`PartNum`	Optional. A user-specified, item-level number that indicates the part number in a multi-part refresh.
`ItemGroup`	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
`PermissionData`	Optional. Specifies the permission information associated with content of this stream.
`NameType`	Optional. This should match nameType specified in the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	**Required**. This should match the requested name.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The application should set either the service's ServiceName or ServiceId but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The application should set either the service's ServiceId or ServiceName but not both.
Payload	**Required**. This should consist of a FieldList containing all fields associated with the item. Some FieldEntrys 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 62: YieldCurve Refresh Message

10.2.3 YieldCurve Update Message

A YieldCurve Update Message is encoded using UpdateMsg by OMM provider and OMM 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 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 Update response messages.

COMPONENT	DESCRIPTION / VALUE
DomainType	**Required**. **MMT_YIELD_CURVE = 22**
UpdateTypeNum	**Required**. Indicates the general content of the updates. Typically sent as one of the following: • **INSTRUMENT_UPDATE_UNSPECIFIED=0** • **INSTRUMENT_UPDATE_QUOTE=1**
Indications	Optional: • DoNotCache **true**: Requests the application to not cache this update message. • ClearCache **true**: Requests that the application clear the cache. • DoNotConflate **true**: Requests the application to not conflate the update message.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
PartNum	Not used.
ConflatedCount	Optional. If a provider is sending a conflated update, this informs the consumer of 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 is sending a conflated update, this notifies the consumer of the times 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. Permissioning information associated with only the contents of this update.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.

COMPONENT	DESCRIPTION / VALUE
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the name of the service that provides the data. **Note:** The application should set either the service's ServiceName or ServiceId but not both.
ServiceId	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The application should set either the service's ServiceName or ServiceID but not both.
Payload	**Required**. This should consist of a FieldList containing all fields associated with the item. Some FieldEntrys 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 63: YieldCurve Update Message

10.2.4 YieldCurve Status Message

A **YieldCurve status** message is encoded using StatusMsg and sent by OMM 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 if **MsgKeyInUpdates** is set to **true**, then the Name and ServiceName must be provided for Status response messages.

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 clear cache
QoS	Optional.
ItemGroup	Optional. The provider can use this component to change the item's ItemGroup.
PermissionData	Optional. If present, this is the new permission information associated with all the contents on the stream.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the name of the service that provides the data. **Note:** The application should set either the service's ServiceName or ServiceId but not both.
ServiceId	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The application should set either the service's ServiceName or ServiceId but not both.

Table 64: YieldCurve Status Message

10.2.5 YieldCurve Domain Post Message

If the provider claims to support this type of posting, consumer applications can post Yield Curve data. For more information on posting, refer to the *EMA 3.0.4 C++ Edition Developers Guide*.

10.3 Data: The Response Message Payload

The payload of a Yield Curve Refresh or Update is an `FieldList`. Some `FieldEntry` contents contain primitive type information to help describe the curve. Some examples of this are the Curve Type (**CRV_TYPE**), the Algorithm (**CRV_ALGTHM**) used to calculate the curve, and the Interpolation (**INTER_MTHD**) and Extrapolation (**EXTRP_MTHD**) methods. Since the fields in each `Vector` are the same, bandwidth can be further optimized by DataDefinitions.

Other `FieldEntry` contents contain more complex information. The more complex entries are broken down into:

- Input Entries which define the different input information 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 Information (**EX_MET_DAT**) which provides general information about the yield curve. This is represented using an `ElementList` type. The extra meta information allows users to provide additional curve description without requiring new fields to be defined. Some examples of this meta information would be the curve creation time or curve's owner.

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**)

Each `VectorEntry` houses an `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, **Decoding FieldList Contents with Field and Enumerated Types Dictionaries**.
- For more information about use of the `Vector` and `FieldList` container typesSee the *EMA 3.0.4 C++ Edition Developers Guide*.

The following table contains additional information on input and output entries.

NAME FIELD NAME TYPE	CONTAINER TYPE	CONTAINER ENTRY TYPE	REQUIRED	DESCRIPTION
Cash Rates CASH_RATES *Input*	Vector	FieldList	No	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*	Vector	FieldList	No	Contains future pricing data used to calculate the yield curve output. This typically includes information like settlement date (**FUTR_SDATE**), maturity date (**FUTR_MDATE**), and basis (**FUTR_BASIS**).
Swap Rates SWAP_RATES *Input*	Vector	FieldList	No	Contains swap rate data used to calculate the yield curve output. This typically includes information like settlement date (**SWAP_SDATE**), maturity date (**SWAP_MDATE**), swap rate value (**SWAP_RATE_VAL**), and roll date (**SWAP_RDATE**).
Spread Rates SPRD_RATES *Input*	Vector	FieldList	No	Contains spread rate data used to calculate the yield curve output. This typically includes information like spread frequency (**SPRD_FREQ**), maturity date (**SPRD_MDATE**), spread rate (**SPRD_RATE**), and roll date (**SPRD_RDATE**).
Yield Curve YLD_CURVE *Output*	Vector	FieldList	No	Contains calculated Yield Curve data. This typically includes information like zero rate (**YCT_ZRATE**), forward rate (**YCT_FWRATE**), and discount factor (**YCT_DISFAC**).

Table 65: Yield Curve Inputs and Outputs

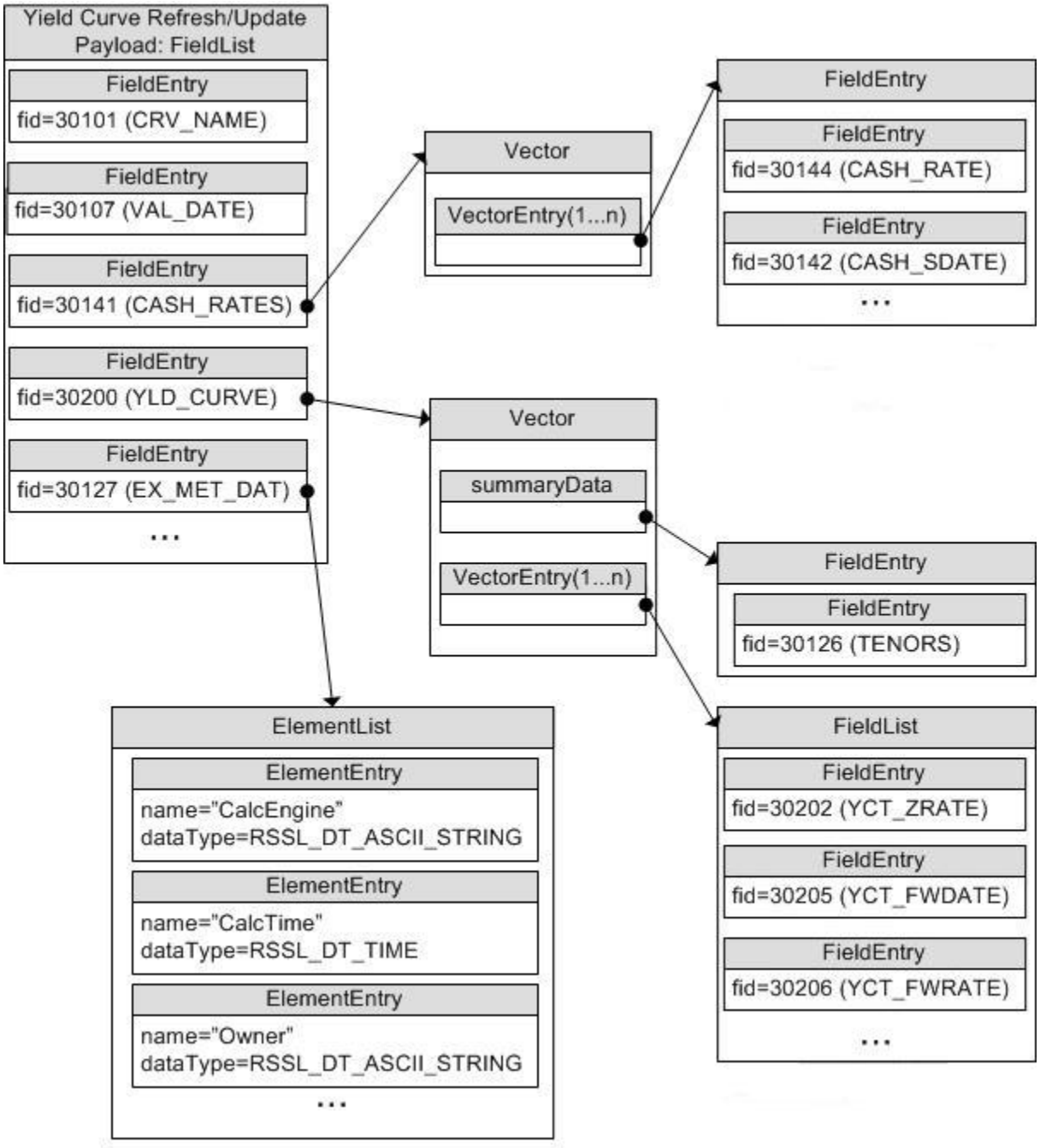

Figure 20: Yield Curve Payload Example

10.4 Special Semantics

None

10.5 Specific Usage: ATS

Please refer to the ATS documentation as there are required FIDs that will need to be in the dictionary used by the application consuming Yield Curve data.

Chapter 11 SymbolList Domain

11.1 Description

The ***SymbolList*** 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` which contains additional cross-reference information such as permission information, name type, or other venue-specific contents.

11.2 Usage

Note: Generic Message(s) are not supported for the **MMT_SYMBOL_LIST** Reuters Domain Model.

11.2.1 SymbolList Request Message

A ***SymbolList*** request message is encoded and sent by OMM consumer applications.

The consumer can make a "streaming" request (set `ReqMsg.InterestAfterRefresh` with **true** value) to receive updates, typically associated with item additions or removals from the list.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_SYMBOL_LIST = 10**
`Interactions`	**Required**. • `InitialImage` **true**: indicates initial image is required • `InterestAfterRefresh` **true**: indicates streaming request is required • `Pause` **true**: indicates pause is required
`Indications`	Optional. • `ConflateInUpdates` **true**: indicates conflated updates is required Batch and View request are specified in the `Payload`.
`Priority`	Optional. Indicates class and count associated with stream priority.
`NameType`	Optional. When consuming from Thomson Reuters sources, typically set to **INSTRUMENT_NAME_RIC = 1** (the "Reuters Instrument Code"). If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	**Required**. The name of requested item. **Note:** Not used for Batch Item request.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. **Note:** The consumer application should set either the service's ServiceName or ServiceId but not both.
ServiceId	**Required**. **Note:** The consumer application should set either the service's ServiceId or ServiceName but not both.
Payload	Optional. When features such as View or Batch or symbol list enhancements are leveraged, the payload can contain information relevant to that feature. For more detailed information about View or Symbol List Enhancements see Chapter 12.

Table 66: SymbolList Request Message

11.2.2 SymbolList Refresh Message

A ***SymbolList*** refresh Message is encoded using `RefreshMsg` and sent by OMM Provider and OMM non-interactive provider applications. This message sends a list of item names to the consumer.

A SymbolList 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 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_SYMBOL_LIST = 10**
`State`	**Required**. Includes the state of the stream and data.
`Solicited`	**Required.** **true**: Indicates the message is a solicited refresh. **false**: Indicate the message is an unsolicited refresh.
`Indications`	Optional: • `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.
`QoS`	Optional. If sent, specifies the quality of service which the stream is provided.
`SeqNum`	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
`PartNum`	Optional. A user-specified, item-level number that indicates the part number in a multi-part refresh.
`ItemGroup`	Optional. Associate the item with an Item Group (refer to Section 4.3.1.3).
`PermissionData`	Optional. Specifies the permission information associated with content of this stream.
`NameType`	Optional. This should match name type specified in the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	**Required**. This should match the requested name.

COMPONENT	DESCRIPTION / VALUE
ServiceName	**Required**. This should be the Name associated with the service from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
ServiceId	**Required**. This should be the ID associated with the service from which the cosumer wishes to request the item. **Note:** The provider application should set either the ServiceId or ServiceName of the service, but not both.
Payload	**Required**. The payload contains a Map, where entry represents an item in the list. Each map entry contains a FieldList with additional information about that item.

Table 67: SymbolList Refresh Message

11.2.3 SymbolList Update Message

A ***SymbolList*** update Message is encoded using UpdateMsg and sent by OMM Provider and OMM 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 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_SYMBOL_LIST = 10**
Indications	Optional. • DoNotCache **true**: Indicates do not cache this update message • ClearCache **true**: indicates clear cache • DoNotConflate **true**: indicates do not conflate this update message
QoS	Optional.
SeqNum	Optional. A user-specified, item-level sequence number which can be used by the application for sequencing messages with this stream.
ConflatedCount	Optional. If a provider is sending a conflated update, this informs the consumer of 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 is sending a conflated update, this notifies the consumer of the times 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. Permissioning information associated with only the contents of this update.
NameType	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, NameType defaults to **INSTRUMENT_NAME_RIC = 1**.
Name	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
ServiceName	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be 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.

COMPONENT	DESCRIPTION / VALUE
ServiceId	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the ServiceName or ServiceId of the service, but not both.
Payload	**Required**. The payload contains a Map, where entry represents an item in the list. Each map entry contains a FieldList with additional information about that item.

Table 68: SymbolList Update Message

11.2.4 SymbolList Status Message

A ***SymbolList*** status message is encoded using `StatusMsg` and sent by OMM 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 if **MsgKeyInUpdates** is set to **true**, then the `Name` and `ServiceName` must be provided for **every** Status response messages.

COMPONENT	DESCRIPTION / VALUE
`DomainType`	**Required**. **MMT_SYMBOL_LIST = 10**
`State`	Optional. Current state information associated with the data and stream.
`Indications`	Optional: • `ClearCache` **true**: Requests the application to clear the cache.
`QoS`	Optional.
`ItemGroup`	Optional. The provider can use this component to change the item's `ItemGroup`
`PermissionData`	Optional. If present, this is the new permission information associated with all the contents on the stream.
`NameType`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name type specified on the request. If it is not specified, `NameType` defaults to **INSTRUMENT_NAME_RIC = 1**.
`Name`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should match the name of the item being provided.
`ServiceName`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the name of the service that provides the data. **Note:** The provider application should set either the service's `ServiceName` or `ServiceId` but not both.
`ServiceId`	Optional (**Required** if **MsgKeyInUpdates** is set to **true**). This should be the ID of the service that provides the data. **Note:** The provider application should set either the `ServiceName` or `ServiceId` of the service, but not both.

Table 69: SymbolList Status Message

11.3 Data: Response Message Payload

The payload is a `Map`. Each `MapEntry` key is an AsciiString symbol. The `MapEntry` value can be empty. It also may be a `FieldList` that includes permission data and cross-reference information. This information should not update frequently.

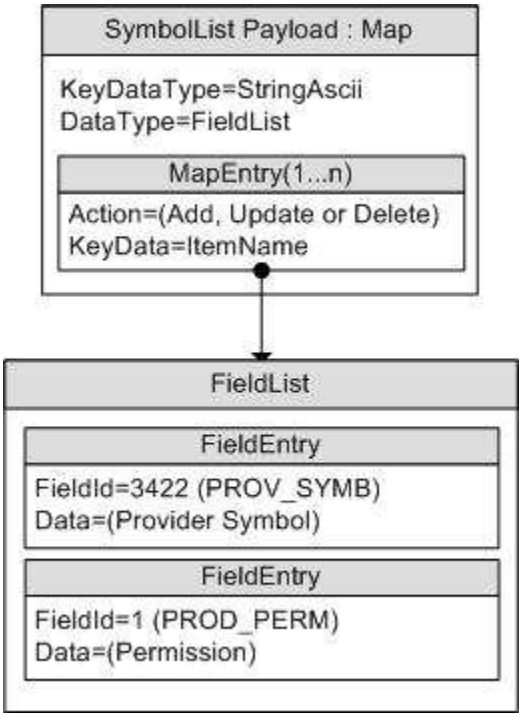

Figure 21: SymbolList Response Message Payload

Fields typically included in a `FieldList` are:

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

11.4 Special Semantics

None

11.5 Specific Usage: RDF Direct

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#`. See the *RDF Direct OPRA Venue Guide* for details.

Chapter 12 Payload in ReqMsg

12.1 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 RDM and any user defined DMM. For more information, refer to the *EMA 3.0.4 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:

ELEMENTNAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLES	DESCRIPTION
:ViewType	UInt	Conditional (see Description)	VT_FIELD_ID_LIST	1 = VT_FIELD_ID_LIST 2 = VT_ELEMENT_NAME_LIST	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**.
:ViewData	Array of Int or Array of ASCII	Yes	No	Array of desired entries. Content matches the type as specified by ViewType e.g., a **ViewType** of **VT_FIELD_ID_LIST** uses an array of field IDs	Field Ids will be encoded as an array of 2 byte fixed length field identifiers. Element names will be variable length Ascii string fields.

Table 70: View Definition in Payload

12.2 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 RDM and any user defined DMM. For further details, refer to *EMA 3.0.4 C++ Edition Developers Guide*.

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

ElementName	TYpe	Required	Default	Range / Examples	Description
:ItemList	Array of ASCII	Yes	No	1 to Array max	A list of item names in which the client registers interest.

Table 71: ItemList in Payload

12.3 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 the client application need not open individual items belonging to symbol list and the items will be opened and data will be provided. To do this, encode request message payload with an element list having an element entry specifying the symbol list behaviour. For further details, refer to *EMA 3.0.4 C++ Edition Developers Guide*.

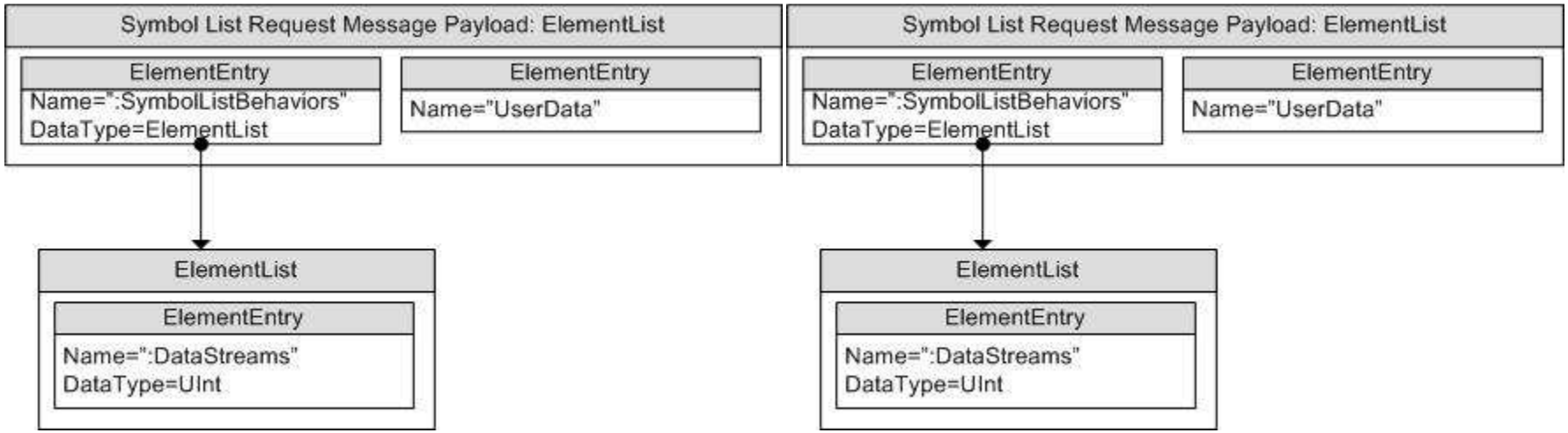

Figure 22: SymbolList Request Message Payload Specifying Symbol List Behavior

For Symbol List request messages specifying interest in data, the payload contains at a minimum an element list which includes the following element entry:

ElementName	Type	Required	Default	Extensible Contents	Description
:SymbolListBehaviors	ElementList	No	ElementList containing ElementEntry of **DataStreams** set to **0**.	Yes	Indication of the expected data behavior of the individual items that will be opened from the symbol list. If this element is not present, individual streams will not be open.

Table 72: Request Message Payload for Symbol List Domain Specifying Symbol List Behaviors

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

ELEMENTNAME	TYPE	REQUIRED	DEFAULT	RANGE/EXAMPLES	DESCRIPTION
:DataStreams	UInt	No	0	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 *EMA 3.0.4 C++ Edition Developers Guide*. These are bit-masks: • **0x0** (or if absent): The consumer is interested only in getting the names and no data on the individual items of the symbol list. • **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 72: ":SymbolListBehaviors" ElementEntry Contents

Document ID: EMAC304UMRDM.160
Date of issue: 19 August 2016

THOMSON REUTERS