Reuters
Open Message Model Technical White Paper

Version 1.0, August 2006

REUTERS

Open Message Model

Contents

Figures

1. Introduction

1.1. Purpose

This white paper focuses on the Open Message Model (OMM) and how it can be used, both by Reuters and other parties, to model many different types of content. OMM is a key component of the overall Reuters Data Model Architecture (RDMA). This architecture defines a layered model which is used to describe and distribute all content. Reuters Data Model Architecture and OMM have been specifically designed to overcome existing limits and to provide a more flexible and efficient basis for exchanging financial data in the future.

Current interfaces and systems provided by Reuters support a range of different data models, most of which have been created around the popular Marketfeed record format. Nearly all applications understand the Marketfeed record format, Unfortunately, Marketfeed is not flexible enough to describe complex data efficiently. This led developers to force some data models into the simple Marketfeed record format, resulting in unwanted complexity and possible inefficiencies. This has also created a generation of applications that use proprietary data formats whose data models used are not available to Reuters products (such as Reuters 3000 Xtra) or other applications.

1.2. Executive summary

Reuters has created a completely new architecture – Reuters Data Model Architecture (RDMA) – for structuring data within RMDS (Reuters Market Data Systems). This will enable applications and systems to send and receive data more efficiently, and will allow application developers the flexibility to access new data types and to model additional data types that meet changing market requirements.

At the same time, Reuters has been careful to ensure continued support for the thousands of custom applications and third-party products that were developed for the old architecture. All key interfaces necessary to run legacy TIB and Triarch applications have been maintained in the new architecture.

1.2.1. Driving change

Recent years have seen a strong increase in the demand for market data. Trading applications, particularly the fast-increasing number of algorithmic-trading applications, are driving this trend. These applications need to scan and analyse huge volumes of data in milliseconds to spot trading opportunities and trigger orders or execution.

In addition to the sheer volume of data, these applications require programmatic access to a broader array of content than ever before, such as level II data (order book, market-maker data), deep tick history, news, portfolios, and even transaction messages. As this trend for new data types to feed applications accelerates, there will be a constant requirement for programmatic access to new data types.

The existing Reuters data structures have served the markets well for more than a decade now, but it has become clear that they are not going to be able to supply the levels of flexibility and performance (both processing speed and throughput) that will become the norm in the future. Consequently, Reuters has rearchitected its data model from top to bottom, in order to create new structures that will meet these future needs, while providing continued support for its existing data structures.

To address all these requirements, the new Reuters data model breaks down into three main areas.

- At the bottom of the model is the Reuters Wire Format (RWF). This is the binary format that is used to express all data. In the past, there has been a number of different formats. Now all these will be replaced by one single format that is more efficient in the way it handles data. This is a key quality, leading directly to shorter message lengths and higher throughput.
- The middle layer is the Open Message Model. This has two important components: a new **transport protocol**, RSSL, which includes new request types (such as those for handling streaming data), and a set of **primitive data structures** (Data Abstractions) that are the building blocks for more complex data types.
- The top layer is the Domain Model. This is where the primitive data types and the relevant transport details are combined to create data types that match market needs. Reuters has already created the standard types that meet existing needs, such as Login, Dictionary, Market Price, Symbol List, and Market-by-Order.

The Domain Model types have been created by composition of the basic OMM message structures such as:

- **Field List.** A list of field/value pairs that replaces the existing logical record.
- **Vector.** A simple integer-indexed (zero-based) vector of information.
- **Map.** A key-indexed vector of information, similar to a hash table or STL map.
- **Series.** A structured table of information (typically used for historical data).
- **Element List.** A list of self-describing field/value pairs (that does not need a dictionary).

1.2.2. Unleashing innovation

The Open Message Model is truly open. Your developers can extend these data types to suit your needs or create completely new ones. This degree of flexibility is a key requirement as applications need to process new types of data, increasing depth of data and new types of transactions.

Additionally, the definition of data structures is no longer dependent on the underlying infrastructure or API upgrades. With the legacy data formats, you have to wait for the new release of the infrastructure before you can exploit any changes to the data structures. With OMM, you design the type definition, code the new type into your application using the RFA, and then deploy it on RMDS very rapidly, potentially in no more than a couple of hours. The only limitation is the imagination of the application developer.

It has always been central to the success of RMDS that development partners and individual institutions were able to leverage the system in different ways to deliver extra dimensions of functionality and value. The flexibility and underlying performance of the new architecture will open up many new opportunities for creating innovative financial applications. In areas like algorithmic trading, OMM is being used to represent new information like full order books, complex yield curves, machine readable news, tick histories, corporate actions, and reference data. In the future, OMM will also be used to deliver new functionality to RMDS applications like transactions and settlements, leveraging standards like FIX and SWIFT in OMM. Messaging is the foundation of the RMDS system, and the switch to flexible data modelling will result in numerous examples like these, heralding a new era of inventiveness.

Extending the concept of openness, the Reuters Customer Zone will be used to share innovations. Recently created data types will be available for download. There will be common-interest groups where different organizations can work together. This will ensure rapid development of, for example, a new transaction type or trade blotter.

1.2.3. Built-in benefits

Market Data Managers will appreciate the efficiencies that are inherent in the new structures:

- The underlying RMDS architecture can be structured to deliver both efficiency and low latency. Using OMM, sub-millisecond latency can be achieved at higher update rates than ever before.
- The binary efficiency of the Reuters Wire Format (RWF) means that message sizes are reduced. Data in OMM is half the size of the same data in current MarketFeed format, and about one-sixth of the size it would be in TIB.
- When using OMM, higher throughput rates can be achieved using RMDS 6 components when compared to RMDS 5 components using MarketFeed. RMDS 6 and OMM can deliver anywhere from 45% to 100% more throughput.
- The binary format of RWF ensures the faster processing speeds that algorithmic applications require. With the data already in binary, there is no longer any need for conversion of strings.
- Equally, the new data structures are inherently faster to utilize in applications because there is no longer any need to parse each message in full. Relationships are defined within the data types, so applications can go straight to the relevant point in the message for the required data.
- The architecture will enable much more effective sharing of data between the front, middle and back offices.

1.2.4. Upgrade with ease

The RMDS 6 was designed to facilitate an easy upgrade. The RMDS 6 components can be operated in conjunction with existing RMDS 5 installations. You can add a Point-to-Point Server (P2PS), which can work simultaneously with both the RMDS 5 RRMP (Restricted Reliable Multicast Protocol) 4 backbone protocol and the RMDS 6 RRMP6 backbone protocol. You can then add new Source Distributors and new OMM-based source applications as they are required. This allows you to meet the needs of new programmatic applications with needs for new data types by simply extending your RMDS. This also allows these new applications to access all of your internally published data, There is no need for wholesale system-change.

Applications using current APIs and data structures can connect to the upgraded infrastructure without any changes, and will continue receiving the information they receive today. The RMDS 6 components can provide automatic conversion of OMM Market Price models (see section 5.1.2) to formats required by existing APIs like SSL, SFC, and the TIB API.

1.3. Reuters Data Model Architecture overview

With RFA/RMDS 6, content will be viewed in a more layered/structured manner as depicted in Figure 1. This structure enables each layer to build on the facilities provided by those beneath and offers a vocabulary for discussing the capabilities. Starting with version 6, the RFA and RMDS components will strictly follow this data model architecture.

Figure 1 – Reuters Data Model Architecture

The three key layers are shown in yellow. From the bottom upwards, these are:

- **Wire Format** – The encoded wire format that is used for distribution between any APIs and/or core components (e.g. RMDS, RDF, RDF Direct). The Reuters Wire Format (RWF) makes up this layer and offers a flexible, bandwidth and CPU efficient binary wire representation that can be used for the distribution, creation and manipulation of all data content.
- **Open Message Model** – OMM provides the constructs for expressing all data content through transport and data abstractions. It is the layer that is implemented in all messaging APIs and core components. The flexibility provided by OMM means that new data models can be created in the future without any need for new versions of the messaging APIs (i.e. RFA) and/or infrastructure (i.e. RMDS). OMM is implemented within the RFA 6 interface (using a new RSSL protocol) and the RMDS 6 infrastructure.
- **Domain Message Model** – The Domain Message Model uses the capabilities of OMM for defining Item Type Models and the Content Definition Model. Item Type Models define real-world types (e.g. Market Price, Market-by-Order, News, Historical, etc.) and represent the realization of select domains using OMM. The actual field meanings/relationships, required fields, instrument types, etc. are defined by the Content Definition Model. Reuters versions of the Domain Message Models are defined by the Reuters Domain Models and should be used to guarantee maximum interoperability.

The Reuters Data Model Architecture will be used to define how all Reuters content is described, structured, accessed and produced through client facing interfaces. Applications supporting these "standards" will inter-operate independent of the vendor/exchange providing the data. Any interested parties can define their own Domain Message Models for item types, and these can be used without software enhancements to RFA or RMDS.

In other words, the Reuters Data Model Architecture allows content to be defined in terms that are familiar to financial professionals and make sense for the data itself. It simultaneously defines extensible and re-usable abstractions that make sense at a systems/messaging perspective. These abstractions allow new types of data to be defined without new releases of RFA or RMDS.

1.4. Scope

This paper discusses the Reuters Data Model Architecture in its entirety.

Section 3 gives an overview of wire formats; the present range of formats will be replaced by one single format for all messages.

Section 4 looks in detail at the Open Message Model and the message types and abstract data types that are the building blocks for defining data.

Section 5 describes the Domain Message Model, where data types from the OMM are used to define real data, such Market by Price. Some of the pre-defined Reuters Domain Models are examined in detail to show how the different elements are used in practice.

The main purpose of this document is to define the OMM architecture. RFA 6 represents the API implementation of this architecture and may therefore use different terminology or expose slightly different attributes (e.g. service name instead of service identifier).

<table>
<tr><td rowspan="2">Domain Message Model</td><td>Content Definition Model</td><td>Field Meanings
Field Relationships</td><td rowspan="2">Reuters Domain Models (RDM)</td></tr>
<tr><td>Item Type Model</td><td>Real World Objects
(i.e. Quotes, Order Books, etc)</td></tr>
<tr><td rowspan="2">Open Message Model</td><td>Data</td><td>Data Containers
Primitive Structures</td><td>RFA 6.0 RSSL Protocol
Data Package</td></tr>
<tr><td>Transport</td><td>Interaction Paradigms
Event Model
Symbology
QoS
Entitlements</td><td>RFA 6.0 RSSL Protocol
Message Package
Session Package</td></tr>
<tr><td colspan="2">Wire Format</td><td>Wire Encoding</td><td>Reuters Wire Format (RWF)</td></tr>
</table>

Figure 1 – Reuters Data Model Architecture

1.5. Glossary

API	Application Programming Interface
Decoding	The act of parsing and converting content from a Wire Format, or external representation, into a machine readable form (e.g. RWF parsing, Marketfeed parsing, XML parsing).
Event Stream	The result of a request/response with interest interaction. When a application requests streaming news headlines, an event stream is created that provides asynchronous headlines (updates) and state information.
HTTP	Hypertext Transfer Protocol
Market-by-Order	Instrument market information sorted by order (i.e. full depth order book – level II content).
Market-by-Price	Top of book instrument market information sorted by best price (i.e. market depth – level II content).
Market Price	Trades, Quotes and inside top of book quotes (i.e. level I content).
OMM	Open Message Model
Reuters Data Model Architecture	Reuters Data Model Architecture
RDF	Reuters Data Feed
RDF Direct	Reuters Data Feed Direct
RIC	Reuters Instrument Code
RFA	Reuters Foundation API. Strategic, low-level, thread-aware messaging API that applications can use to publish and consume OMM.
RMDS	Reuters Market Data System
RSSL	Reuters SSL Protocol
RWF	Reuters Wire Format
SFC	System Foundation Classes
Transformation	The act of changing the fundamental structure of the data/model from one to another (e.g. RWF to Marketfeed conversion, Marketfeed to TibMsg self-describing conversion).
Wire Format	An encoded, hardware neutral, external representation of a data model that provides for the transmission of the model between multiple components (usually over a LAN or WAN). Wire formats may be optimized using many different networking tricks (e.g. nibbles, bitmaps, integer chains, etc.).

2. Fundamentals

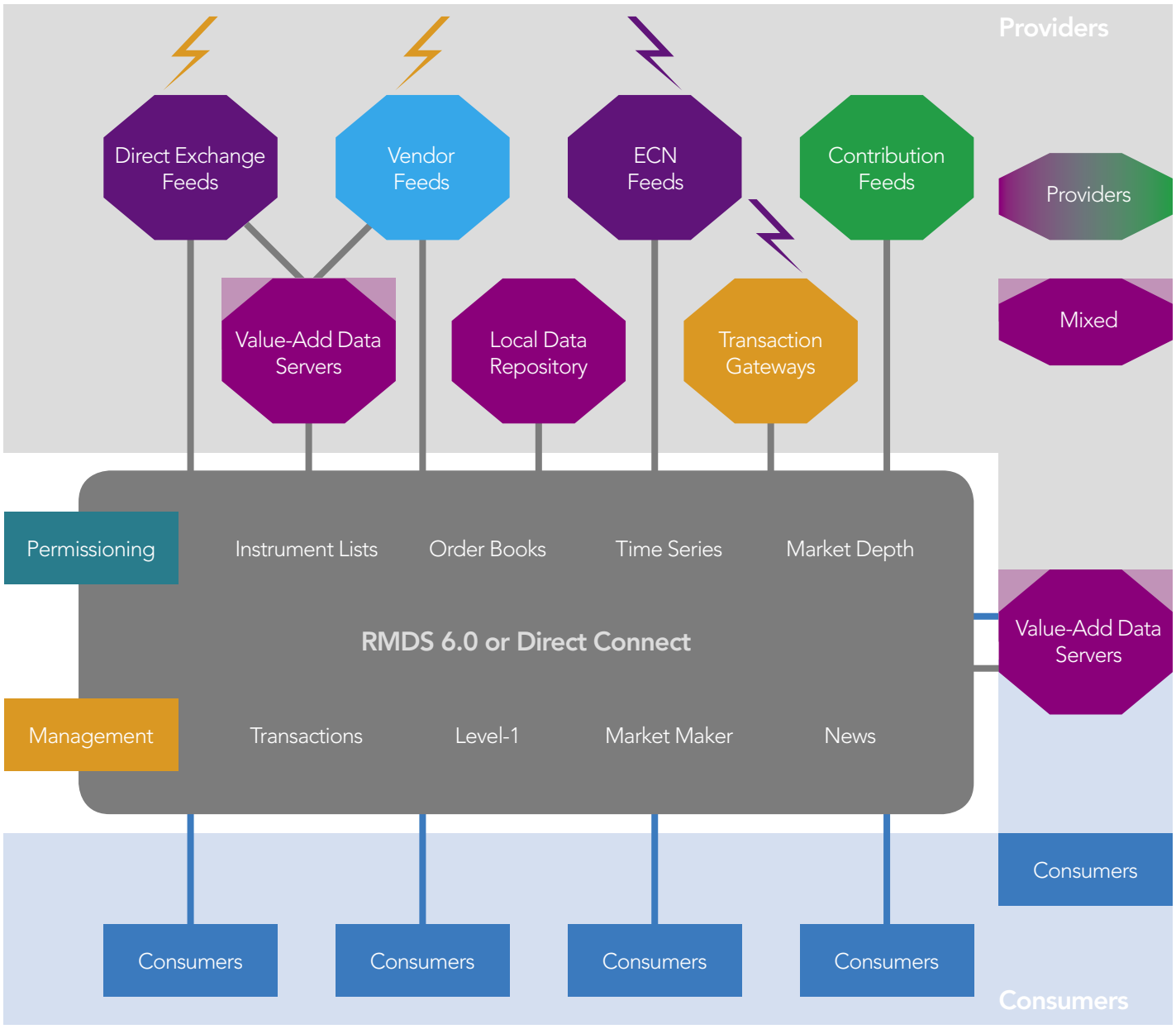
Some key terminology is new to RFA/RMDS 6. The terminology and concepts are introduced here.

2.1. Provider/Consumer Model

Figure 2 – Provider/Consumer Model

The RFA API and RMDS infrastructure are designed to enable information and capabilities to be shared by different users. Users are divided into Providers, who seamlessly make capabilities and information available, and Consumers, who, with the relevant permissions, have access to the capabilities and information. The term Provider, is used instead of publisher, because this type of application may not only publish data (e.g. direct exchange feed, RDF), but it may also provide a capability (e.g. Exchange gateway – transactions, Vendor Contributions) to many different consumer applications. Consumers always interact with a provider, either directly or via RMDS, in order to access the capabilities or data content they require.

Figure 2 – Provider/Consumer Model



2.2. Services

Services provide a unique identification scheme for breaking up different types of providers. They offer a convenient mechanism to manage large sets of data and/or capabilities. All request/response traffic is directed to and received from a service provider. Services can be categorized by many different criteria (e.g. business classification, consolidated vender, direct exchange feed, exchange gateway, etc.).

Services are dynamic in nature and thus may be created or removed on the fly. Their existence and characteristics are detected dynamically by the consumer applications.

2.3. Access Point

Figure 3 – Consumer Access Points
Consumers utilize content and capabilities from providers through access points. Consumer access points currently manifest themselves in two different ways:

- Direct. A provider that supports direct connections represents a concrete consumer access point. Multiple consumers can directly interact with the provider in order to access the content and capabilities offered by the service provider. Datafeeds (e.g. RDF, RDF Direct) or exchange gateways typically implement this style of access point.
- Proxy. When certain capabilities are required (e.g. multiple content providers, large scale distribution, local content management, resiliency, etc.), RMDS components can be placed between the concrete providers and the consumers. In this case consumers interact with RMDS components acting as proxies to the concrete service providers. The point-to-point server, current RTIC and future multicast server implement this style of access point.

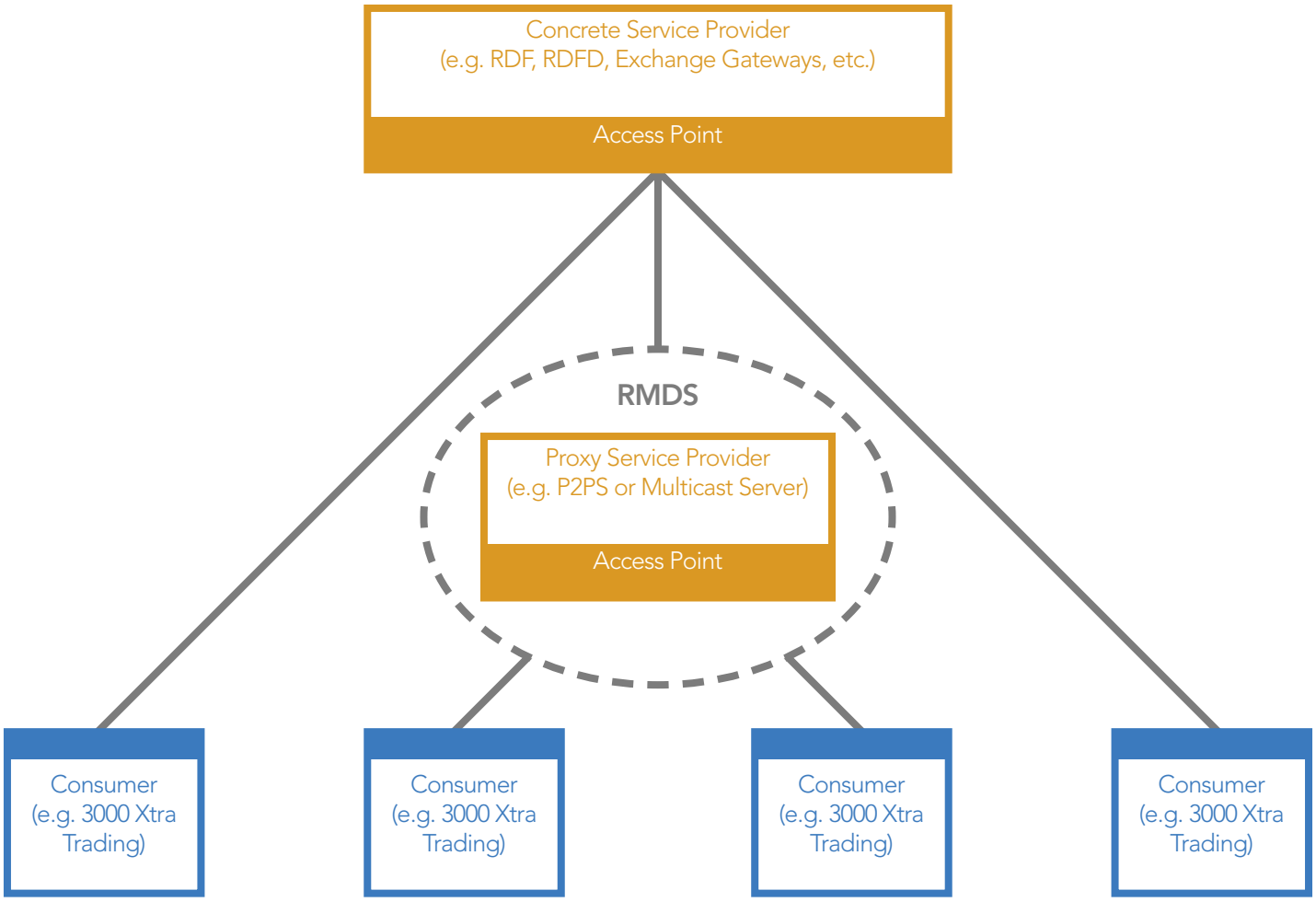


Figure 3 – Consumer Access Points

3. Wire Format

A wire format defines a hardware neutral, external data encoding for transmitting data between multiple components/machines. The components can reside on the same or different machines or even on different machine architectures (i.e. Win32-x86, Linux-x86, Solaris-SPARC, Solaris-x64). Most network-based communications happen over a Local Area Network (LAN) or a Wide Area Network (WAN). Many different wire formats exist that allow the transferring of simple data types (e.g. integers, floats, time) between different parties.

Currently, Reuters APIs and systems define and utilize many different wire formats. In fact, the wire format is usually coupled with the data being transported. Even though many of the wire formats utilize very common techniques, each has its own definition. Examples of current wire formats include:

- **Marketfeed** – a delimited ASCII based wire data format that is used to represent logical records. Field identifiers are used to identify particular fields and require a dictionary to make sense of the data. The Marketfeed format is documented and encoded/decoded within applications.
- **QForms** – a binary based wire data format that is used to represent logical records. Field identifiers are used to identify particular fields and require a dictionary to make sense of the data. The dictionary is also required to parse the message. QForms encoding and decoding are implemented within the TibMsg API.
- **TibMsg** – A self-describing binary based wire format that is used to represent logical records. TibMsg encoded messages are typically much larger than QForms or Marketfeed due to their self-describing nature. TibMsg encoding and decoding are implemented within the TibMsg API.
- **ANSI Page** – A page format that uses ANSI sequences to express the displayable data.
- **SSL** – a point-to-point binary based wire format that is used to transport messages between components. It is currently implemented within the SSL4.x APIs.
- **RRMP** – a broadcast/multicast binary based wire format that is used to transport messages between RMDS components. It is currently implemented in the Source Distributor, P2PS, rtic and rdtic.

3.1. Reuters Wire Format

In RFA6 and RMDS6, these different formats are replaced by a single wire format – Reuters Wire Format (RWF). Irrelevant of whether the layer performs transport behavior, or implements data formats, the Reuters Wire Format will be used to encode the information.

RWF uses binary encoding, bit-wise operations and reserved values in order to optimize all network distribution. RWF is automatically encoded into network byte order for transmission and then decoded into the machine-specific byte order for consumption. When properly combined, the binary nature of RWF provides smaller messages that are easier and more efficient to encode/decode, thus resulting in an increase in total throughput.

The base Reuters Wire Format defines primitive data types or building blocks that can be transmitted between multiple components. The Open Message Model uses these primitive types to represent more complex transport and data formats between components. In effect, OMM uses, and then extends, RWF to build more complex messages types and data formats.

Some of the data type encodings that make up the base Reuters Wire Format include:

- Fixed-sized signed and unsigned 8-bit, 16-bit, 32-bit and 64-bit Integers. All values are encoded in network byte order and use the same number of bytes as defined by the value used (e.g. 16-bit – encoded in 2 bytes). The two's-complement IEEE format is used to encode signed integer values.
- Special value variable sized unsigned 16-bit and 32-bit integers. These values must always contain a single byte. If the byte is less than a reserved value, then that byte is the actual value. Otherwise, the value of the first byte is used to infer the number of bytes the value is encoded after this first byte.
- Reserved bit-variable-sized unsigned 15-bit, 30-bit and 62-bit integers. These values must also always contain a single byte. However, one, or two, high-order bits are reserved in the first byte to indicate the total length of the integer value.
- Real (decimal) values that contain a maximum 32-bit or 64-bit integer coefficient and a 6-bit integer exponent. The two high bits in the exponent byte can be used to identify the length of the exponent for bandwidth optimizations.
- IEEE standard 754 floating-point numbers (floats-4 byte and doubles-8 byte).
- Time (hour,minute,second,millisecond), Date (day,month,year), and combined Date/Time.
- Multiple buffers that have different maximum lengths depending on the integer encoding used.
- ASCII String, RMTES String and UTF8 String.
- Array of any of the above data types.

4. Open Message Model

The Open Message Model forms the basis for all messaging starting with the 6.0 release of RFA and RMDS. It does this through re-usable transport and data abstractions. All current and future data content will be described, structured, accessed and produced using the OMM concepts. Also, most new data models can be realized without modifications to the RFA or RMDS components. Low latency and high throughput of messages was a main design goal of OMM. Abstractions have been carefully created in order to properly weigh performance with overall flexibility. When these goals came in direct conflict, multiple options have been employed to maintain performance were necessary.

Previous interfaces and RMDS components are divided into transport level semantics and data format semantics. The transport level (e.g. SSL4.X/SASS3/RRMP) supports multiple services that provide generic item request, response, update and status semantics. Items are identified with a unique name (i.e. buffer and length) within a particular service. The items that are provided through the transport carry many different data formats (including user/client defined). However, the most popular utilized data formats are logical records and pages. Logical records define everything as a combination of field/value pairs while pages are used more for display.

When looking at many different types of information and how that information is accessed, many similarities can be identified. These commonalities can be found in both how you access the data and what form the data itself takes. The request/response with interest pattern that is used to access Market Price (level-1) content can be used to access all "streaming" content (i.e. open ended requests). The main difference is meanings between the responses and any event stream updates. Some examples of this concept include:

- Market-by-Order, or Order Book, information can be accessed using the request/response with interest pattern. The response to the request contains the actual order book information that is structured in a common data format. Updates represent modifications to the book received in the response and need to be applied appropriately.
- Streaming News Headlines can also be accessed using the request/response with interest pattern. The response acknowledges the request and does not actually contain any information. Updates represent actual asynchronous headlines that are structured in a common format. Updates also contain an extra permission expression that must be checked against the user's profile.

The OMM design approach uses generic messages, attributes and data formats that defer specific semantics to the Domain Message Model. Data representations are also abstracted and separated from the behavior in order to offer the most flexible model. Applications can provide and consume any content using the transport and data abstractions defined within OMM.

The OMM is divided into two different layers, each of which is described in detail in this section:

- The Transport layer defines the interaction that can take place between a consumer and a provider, such as a request for information or a request to refresh an existing set of information. The Transport Layer is described in section 4.1.
- The Data Abstractions are the generic formats for data, such as the Field List, Vector, etc. These generic formats can be combined (e.g. a vector of field lists or a vector of vectors) to create formats that suit different types of real data, such as an order book. The Data Abstractions are described in section 4.2.

4.1. Transport Layer

The OMM transport layer encapsulates all messaging syntax and semantics. It defines generic messages and attributes within these messages, which defer actual meanings to the Item Type Models. Generic interaction paradigms are combined with an extensible item identification mechanism (i.e. key) in order to provide the most flexibility. Built-in state information exists for both the data and any event streams. A request priority scheme exists that can be used to prioritize request traffic (in terms of both initial request and any possible recovery)[1]. Event streams can be broken into different groups in order to allow for efficient state transitions.

Response and/or update data can optionally contain permission expressions that define any extra requirements needed to see the data. These permissions expressions, tied with the users permissions profile, allow for full control of user access to content. The Login (see 5.1.1) item type model must be used with an access point to authenticate users and retrieve their permissions profiles in order to properly implement permissioning.

The transport supports both message fragmentation/reassembly and multi-part responses for dealing with potentially large data. Message fragmentation and reassembly is automatically handled by the transport, however it blocks any other data until all fragments have been sent. Multi-part responses allow providers to interleave time-sensitive data with large response data (even within the same item). Both concepts need to be used together when properly implementing a provider with large data content.

The transport layer has been designed for both point-to-point and multicast based delivery. However, the first implementation of RFA 6.0 only provides point-to-point delivery. This allows connections to several access points (i.e. API-to-API, API-to-Source Distributor, API-to-P2PS). Multicast and HTTP/S

based implementations of the API will be made available in future releases.[2]

4.1.1. Concepts

This section goes into detail on some of the key concepts used to define the OMM transport layer.

4.1.1.1. Interaction Paradigms

Consumers must work with providers in order to make use of their many different capabilities. The actual behavioral collaboration, or abstract access methods, between these two parties have been classified into three different interaction paradigms.

- **Request/Response** – A consumer requests snapshot data, or a static capability, from a service provider. The provider typically acts on the request and responds appropriately. Responses can optionally contain data and be broken into multiple parts.[3] However once the response is complete, the interaction is complete. Examples include snapshot News Searches, snapshot Market Prices (Level-1) and Time-series data.
- **Request/Response with Interest** – A consumer requests data or capabilities that can change over time from a service provider. Providers act on the request and respond appropriately. However, the interaction remains active (an event stream is created) and asynchronous events are sent to the consumer. These streaming events can represent changes in received data or notifications of new information. Examples include exchange information (Market-by-Order/Market-by-Price/Market Price), News Headlines and Symbol Lists.
- **Listen/Send** – Also known as publish/subscribe. A provider sends data without the knowledge of any possible consumers. Consumers anonymously listen for data without the knowledge of the providers. This interaction paradigm is currently not used in any Reuters Domain Models since all modeled domains require a high level of permissioning and control.

4.1.1.2. Key

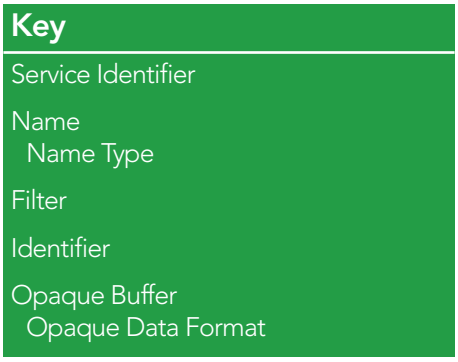


Figure 4 – Key

Access to many different types of data content and capabilities requires an extensible identification mechanism. The key implements the identification mechanism of OMM and replaces the old service name/item name scheme. It is made up of many optional elements that are identified and given meaning by each upper level domain. The current elements that make up the key are:

- **Service Identifier** – an integer representing the service identifier or the service provider. Not all item types, or domains, are directed to specific services (e.g. login).
- **Name** – name of the information requested (consists of a buffer and length). The maximum length of the name is 255 characters and it does not always have to contain printable characters. The name can be used to represent a symbol, a username, or other named types of content.
- **Name Type** – an enumeration representing the different forms the name can take. The name type can be used to represent different symbologies (e.g. RIC, ISIN, CUSIP), multiple username schemes (e.g. email, user identifier), or any other forms the name can take within that domain. The name type enumeration, for certain item type models, can be expanded to include third-party or client defined name types (e.g. internal symbology).
- **Filter** – a bitmap of optional data content/formats logically separated by the provider. The maximum number of filters, or bits, is 32. This high level filtering capability is used to break up different source directory information into request able categories. It is not designed for field level requesting within a field or element list.
- **Identifier** – a simple integer based value for identifying different information. The identifier can be used to represent a version number within some request.
- **Opaque** – an opaque buffer, or extensibility mechanism, allowing a complex identification mechanism (e.g. query, complex filters, etc.). The opaque element can be used to provide an SQL/XML query to a historical database or any other complex parameter lists to a provider.
- **Opaque Data Format** – The data format of the opaque data buffer.

This generic Request Key can be used for generic identification, forwarding and updating of item content. This allows items to be identified/matched using a much more complex scheme than just a name or a subject. Item type models use the key by giving domain specific meaning to the different key elements.

[1] Not supported in RFA 6.0. This will be implemented in a future release.

[2] The RMDS backbone, or Market Data Hub, continues to be based on multicast/broadcast technology.

[3] Single message responses can never receive update messages; however certain domains that use multi-part responses may interleave updates with the response messages.

4.1.1.3. State

State			
Stream State	**Data State**	**Code**	**Text**
Unspecified Open Non-Streaming Closed Closed Recover Redirect	Unspecified Ok Suspect	None Not Found Timeout Not Entitled …	

Figure 5 – State

Generic item level status, or the item condition, is made available through a generic state model. This model represents a normalized view of status and separates stream state from data state. Response status, in terms of both requests and unsolicited responses, is not contained within the generic state model defined within this section. This allows for a clear separation of response state from event streaming state. The elements that make up state include:

- **Stream State** – the state of the event stream when using the request/response with interest paradigm. All non-streaming requests will contain a stream state of non-streaming.
  - **Unspecified** – the state of the stream was not specified or the request is pending.
  - **Open** – the event stream is actively open and asynchronous events can happen at any time.
  - **Non-Streaming** – the information/capability requested does not support streaming semantics or the request was non-streaming. The item will not incur interest after the final response. For example – a consumer can ask for a streaming news search, the provider can respond with the data and a stream state of Non-Streaming.
  - **Closed** – the stream is closed and is not available from the provider at this time. It may be made available in the future.
  - **Closed Recover** – the stream is closed and should be re-opened by the consumer.
  - **Redirected** – the information, or capability, requested is available somewhere else as identified in the key (i.e. similar to an HTTP redirect). The consumer should re-request using the key provided in the response message. Any parameter in the key, including the service identifier, can be re-defined in a redirect.
- **Data State** – represents the quality of the data in the response or in the stream.
  - **Unspecified** – the state of the data is unspecified.
  - **Ok** – the state of the data is ok.
  - **Suspect** – the state of the data is unknown or stale.
- **State Code** – additional status information for the stream or data state. Not needed for generic state processing. Developers may define new state codes when necessary.
  - **None** – no additional information is available.
  - **Not Found** – the item is not available from the provider.
  - **Timeout** – the request has timed-out.
  - **Not Entitled** – the consumer is not entitled to access the item.
  - **Invalid Argument** – an invalid argument was passed in the request.
  - **Usage Error** – illegal usage of messages or message data content.
  - **Preempted** – the event stream has been preempted in order to create room for another event stream.
  - **JIT Conflation Started** – just-in-time, or backpressure based, conflation has started.
  - **Realtime Resumed** – just-in-time conflation has ended.
  - **Failover Started** – source mirroring failover has started on a service.
  - **Failover Completed** – source mirroring failover is complete for a service.
  - **Gap Detected** – A service has detected a message gap from data originator (e.g. exchange).
  - **No Resources** – No more resources exist in order to handle the request.
  - **Too Many Items** – The user has reached the maximum number of event streams available to the application (e.g. as defined by the system administrator).
  - **Already Open** – The event stream is already open for the consumer.
  - **Service Unknown** – The service identifier in the request key does not exist.
  - **Not Open** – The event stream is not open and cannot be closed.
- **Text** – textual information about the stream and/or data state.

4.1.1.4. Quality of Service

Quality of Service	
Timeliness	**Rate**
Realtime Delayed *Delay Time* Closed	Tick By Tick Time Conflated *Conflated Time* JIT Conflated

Figure 6 – Quality of Service

Domains may classify data/events in order to provide differentiated tiers of service. Quality of Service provides this classification and is divided into orthogonal sets of distinct properties. The two properties that make up QoS include:

- **Timeliness** – age of the data
- **Rate** – maximum period of change in data (for streaming events)

Timeliness of data has been broken into the concepts of *real-time* or *delayed*. Real-time implies no delay is applied to the data (it is up-to-date and sent by the provider as soon as it happens). Delayed implies a view of the data in the past and usually includes the delay time.

Rate of change can be categorized as *tick-by-tick*, *time conflated* or *just-in-time conflated*. Tick-by-tick implies the consumer receives every update, or change, in the data. Data is conflated when multiple events are combined in a way that preserves the final view of the content. Conflation can be based on time or vary based on more complex parameters (e.g. channel capacity, congestion, etc.).

Quality of service always contains a single value in each dimension (e.g. Realtime/TickByTick, Realtime/TimeConflated). Consumers can specify QoS parameters on requests and providers will identify QoS attributes for all data and event streams. Note that not all domains utilize the Quality of Service concepts.

4.1.1.5. Group Identifier

Item Groups are used to efficiently update the state of many event streams that originate from a single Provider. Using a single status to report that an entire item group has become stale is much more efficient than using a Status for each of the separate event streams.

Each open event stream belongs to an item group as defined by the group identifier. The item group association is set by the Provider in the initial Refresh. The item group can be modified by the Provider with a Status message or another Refresh message. Providers can establish item groupings on any basis that makes sense to the application's needs. For example, a Provider that maintains multiple data links to data services might establish an item group for each such link. This would allow the Provider to mark all of the items from a given link as being suspect while not modifying the state of items provided by the other links.

4.1.2. Messages

The transport layer defines the messages, and their semantics, that can flow between Provider and Consumer applications. These symmetric messages are used by both the Consumer and Provider in order to communicate (i.e. Symmetric Messaging Paradigm). This not only avoids redundant messages and methods, but also reduces the learning curve when coding either a Consumer and/or Provider application. Not all of the messages defined in this section are used by all of the Domain Message Models. Each model identifies which messages are used and extends their actual meaning.

Most messages contain data that is hierarchal and extensible. It may be sent from a Provider to a Consumer or from a Consumer to a Provider. Data can take the form of attribute or payload and is given meaning by the Domain Message Models. Attribute data is typically additional message attributes (e.g. state, sequence number) while payload data is information that satisfies some business purpose (e.g. Level II data, Transaction data). See the message definitions in this section for attribute data details and section 4.2 for details on payload data.

4.1.2.1. Base

Base
Type
Stream Identifier *Extended Header*

Figure 7 – Message Base

The messages defined all contain the same base attributes. Optional attributes are italicized.

- **Type** – the item type model represented in this message (e.g. Login, Market Price, News Headlines, etc.).
- **Stream Identifier** – an optimization that allows applications to refer to event streams with an unsigned 32 bit value instead of the full key. This consumer defined value can be sent in updates instead of the key (e.g. service id, item name) in order to save bandwidth. Consumers that want the key placed in every update can optionally request this behavior from the provider.
- **Extended Header** – an optional extension to the message header in case a message attribute is identified that currently doesn't fit into any generic message attributes. The extended header is currently not used by any Reuters Domain Model, however is may be used by any item type model in the future.

4.1.2.2. Request

Request	
Type	Options
Stream Identifier	- Streaming - Key In Update - Conflation Info in Update - No Refresh
Data Format	
Priority	
Extended Header	
Best QoS	
Worst QoS	
Request Key	
Payload Data	

Figure 8 – Request Message

A request message is sent from a Consumer to a Provider when it wants to request some data, or a capability, available from the provider. It can also be used to obtain a new refresh or change selected attributes (e.g. priority) for an already open event stream. The generic attributes, and their generic meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the event stream (see 4.1.2.1). It can also be used to match the request and responses.
- **Options** – specifies some of the different request options available.
- **Streaming** – the application wishes to create an event stream based on this request (i.e. the request/response with interest interaction paradigm).
- **Key In Update** – the consumer wants the key encoded in every update.
- **Conflation Information In Update** – the consumer wants any update conflation (e.g. number of updates conflated) information included in the update.
- **No Refresh** – the consumer is trying to update some simple meta information (e.g. priority) about a previous request, or event stream, and does not want any responses.
- **Data Format** – generic format of the payload data (see 4.2).
- **Priority** – when specified indicates the relative importance of the request/data stream.
- **Extended Header** – request header extensions (see 4.1.2.1).
- **Best Quality of Service** – when specified, indicates the upper bounds of the quality of service required by the application (e.g. application prefers Realtime/TickByTick data but will accept down to the Worst QoS).
- **Worst Quality of Service**[6] – when specified, indicates the lower bounds of the quality of service required by the application. When not specified, the best QoS defines the exact QoS required by the application (e.g. application requires Realtime/TickByTick data).[7]
- **Request Key** – The key (e.g. service identifier, symbol, symbology, etc.) representing the data content or capability requested (see 4.1.1.2).
- **Payload Data** – the actual raw encoded data buffer. The actual data format type is identified by the Data Format attribute above. Current providers typically don't accept payload data in requests. However, new types of providers have been identified that may contain payload data in requests (e.g. transaction gateways will accept transaction requests that actually contain the transaction information in the payload data).

[6] Worst QoS not specified in RFA 6.0 (assumes any QoS lower than the best).

[7] Worst QoS not supported in RMDS 6.0.

4.1.2.3. Refresh

Refresh	
Type	Options
Stream Identifier	- Solicited - Refresh Complete - Trash Cache - Provider Driven
Data Format	
Group Identifier	
Sequence Identifier	
Permissions Expr.	
Extended Header	
State	
QoS	
Request Key	
Payload Data	

Figure 9 – Refresh (Response) Message

The refresh message is used to respond with attribute information/data content for a request or can be used to asynchronously change the data of an already opened event stream (i.e. unsolicited images). The Solicited flags within the options identify whether or not the message is a response or a refresh. The generic attributes, and their meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the event stream (see 4.1.2.1). It can also be used to match the request and responses.
- **Options** – specifies some of the different refresh options available.
  - **Solicited** – indicates whether the message is a solicited response to a request or an unsolicited refresh to an existing event stream.
  - **Refresh Complete** – indicates that the response or unsolicited refresh is complete. Some item type models require a single response that will have this flag set with the data. Others allow multi-part responses that will have this flag set in the last response message.

- **Trash Cache** – an indication that any previous payload data cache for the item needs to be deleted.
- **Do Not Cache** – it does not make sense to keep a last value cache of the payload data in this response (e.g. responses for unique queries like Time-Series data).
- **Provider Driven** – the item, identified in the key, is being sent to the consumer without a request (i.e. broadcast mode). This is currently used internally and will not be received by standard consumers.

- **Data Format** – generic format of the payload data (see 4.2).
- **Group Identifier** – the group identifier of the event stream (see 4.1.1.5).
- **Sequence Number** – when specified, indicates the last sequence number associated with the event stream as received by the true data source (e.g. exchange sequence number). These sequence numbers do not have to be sequential for a single event stream (e.g. market price updates for IBM.N may be non-contiguous).
- **Permissions Expression** – when requested using the Login Item Type, contains the permissions expression needed to access the item. This permissions expression defines the requirements needed to access the item data/event stream.
- **Extended Header** – refresh header extensions (see 4.1.2.1).
- **State** – indicates the stream state and data state for the item (see 4.1.1.3).
- **Quality of Service** – when specified, indicates the actual Quality of Service of the actual response and/or event stream (see 4.1.1.4).
- **Response Key** – The key (e.g. service identifier, symbol, symbology, etc.) representing the data content or capability in the response (see 4.1.1.2). The response key can be different from the request key if the provider supports aliasing (i.e. symbology mapping). For example, a consumer may request market price information for an ISIN, the provider can respond with the data and indicate in the response key that the item is actually referred to as a particular RIC.
- **Payload Data** – the actual raw encoded data buffer. The actual data format type is identified by the Data Format attribute above.

4.1.2.4. Update

Update	
Type	Options
Stream Identifier	- Do Not Cache - Do Not Conflate - Do Not Ripple
Data Format	
Update Type	
Sequence Number	
Permissions Expr.	
Extended Header	
Conflation Info	
Update Key	
Payload Data	

Figure 10 – Update Message

The update message is used to represent asynchronous data events associated with an already opened event stream. Item type models may assign different meaning to updates depending on the domain implemented. The generic attributes, and their meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the event stream (see 4.1.2.1).
- **Options** – specifies some of the different update options available. In some cases update options allow a provider to put some context into the update that in the past had to be inferred by all consuming applications/ devices.
  - **Do Not Cache** – it does not make sense to cache (i.e. last value cache) the payload data in this update (e.g. News Headlines, Indications of Interest, Advertised Trades, etc.).
  - **Do Not Conflate** – the payload data in this particular update should not be conflated (e.g. Trades in the Market Price domain, News Headlines, etc.).
  - **Do Not Ripple** – do not ripple any fields within the update (e.g. level-1 data closing run).
  - **Provider Driven** – the item, identified in the key, is being sent to the consumer without a request (i.e. broadcast mode). This is currently used internally and will not be received by standard consumers.[9]
- **Data Format** – generic format of the payload data (see 4.2).
- **Update Type** – the type of update as defined by the item type model (e.g. Trade, Quote, News event for Market Price content). Update types a represented as an expandable enumeration.
- **Sequence Number** – when specified, indicates the sequence number associated with the event as received by the true data source (e.g. exchange sequence number). These sequence numbers do not have to be sequential for a single event stream (e.g. market price sequence numbers for IBM.N may be non-contiguous).
- **Permissions Expression** – when requested using the Login Item Type, contains the required permissions expression needed to access this particular update message (e.g. News Headlines are separately permissioned). It does not affect the permissions expression used for accessing the entire event stream.
- **Extended Header** – update header extensions (see 4.1.2.1).

[9] Not supported in RFA 6.0. This will be implemented in a future release.

- **Conflation Information** – when requested provides the information about any conflation logic that may have been applied to this event. Current parameters include the number of events conflated and/or the time between the conflated events.
- **Update Key** – when requested, the key (e.g. service identifier, symbol, symbology, etc.) representing the event stream for the update (see 4.1.1.2). If the provider aliased the key in the refresh, then it matches the aliased key. By default the update key is not included in the update.
- **Payload Data** – the actual raw encoded data buffer. The actual data format type is identified by the Data Format attribute above.

4.1.2.5. Status

Status	
Type	Options
Stream Identifier	- Trash Cache - Provider Driven
Group Identifier	
Permissions Expr.	
Extended Header	
State	
Status Key	

Figure 11 – Status Message

The status message is used to represent asynchronous attribute changes associated with an already opened event stream. The generic attributes, and their meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the event stream (see 4.1.2.1).
- **Options** – specifies some of the different status options available.
  - **Trash Cache** – any previous payload data cache for the item needs to be deleted.
  - **Provider Driven** – the item, identified in the key, is being sent to the consumer without a request (i.e. broadcast mode). This is currently used internally and will not be received by standard consumers.[10]
- **Group Identifier** – when present the new group identifier of the event stream (see 4.1.1.5).
- **Permissions Expression** – when asked for using the Login domain, contains the new permissions expression needed to access the event stream (e.g. permissions change for access to an item).
- **Extended Header** – status header extensions (see 4.1.2.1).
- **State** – indicates the new event stream state and data state for the item (see 4.1.1.3).
- **Status Key** – the key (e.g. service identifier/symbol/symbology) identifying the event stream (see 4.1.1.2). If the provider aliased the key in the refresh, then it matches the aliased key.

4.1.2.6. Close[11]

Close	
Type	Options
Stream Identifier	- Ack
Extended Header	

Figure 12 – Close Message

The close message is used to close an outstanding request or an existing event stream. The generic attributes, and their meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the request or event stream to close (see 4.1.2.1).
- **Options** – specifies some of the different close options available.
  - **Ack** – the provider should acknowledge the close when received and applied.
- **Extended Header** – close header extensions (see 4.1.2.1).

4.1.2.7. Acknowledgement[12]

Ack	
Type	Options
Stream Identifier	- IsNak
Ack Identifier	
Nak Code	
Nak Text	
Extended Header	

Figure 13 – Ack/Nak Message

The ack message is used to acknowledge an outstanding request or close. The generic attributes, and their meanings, that make up this message include:

- **Type** – item type model (see 4.1.2.1).
- **Stream Identifier** – integer value representing the request or event stream to acknowledge (see 4.1.2.1).
- **Options** – specifies some of the different acknowledgment options available.
  - **IsNak** – this particular message represents a Nak and contains the Nak code and text.
- **Ack ID** – the acknowledgement identifier.
- **Nak Code** – the Nak code (only set for Nak messages).
- **Nak Text** – the Nak text (only set for Nak messages).
- **Extended Header** – acknowledgment header extensions (see 4.1.2.1).

[10] Not supported in RFA 6.0. This will be implemented in a future release.

[11] Not an explicit message in RFA 6.0 (accomplished by closing the handle).

[12] Not supported in RFA 6.0. This will be implemented in a future release.

4.2. Data Abstractions

OMM provides data abstractions that are used to represent disparate data models and are known by applications and infrastructure components (RMDS, RDF Direct) alike. They include field/value pairs constructs in conjunction with sequential and associative containers. The OMM data abstractions are sufficiently flexible to contain additional data abstractions in an arbitrary nesting hierarchy. This flexibility enables the Domain Messaging Models to realize comprehensive data models on top of the abstractions. The payload data formats contained within the transport messages are defined by the data abstractions/formats.

4.2.1. Concepts

Basic concepts are used in the definition of all data abstractions. This section covers these concepts.

4.2.1.1. Data Types

Basic Data Types are contained within certain data formats and attribute information. They allow for the representation of many different types of data content within the given formats. The current defined data types include:

- Signed Integer values (32 and 64 bit)
- Unsigned Integer values (32 and 64 bit)
- Real values that contain both a coefficient and an exponent (32 and 64 bit coefficient with an exponent from +7 thru -15). Examples include:
  - *25.653 can be represented as a real (coefficient = 25653, exponent = -3)*
  - *-100.01 can be represented as a real (coefficient = -10001, exponent = -2)*
  - *1256000 cam be represented as a real (coefficient = 1256, exponent = 3)*
- Time in the form hours, minutes, seconds and milliseconds
- Date in the form day, month, year
- Combined data and time
- IEEE 754 floating point numbers (32 and 64 bit)
- Enumeration
- Binary Buffer
- ASCII String
- RMTES String – Reuters Multilingual Text Encoding Standard (can contain partial field update semantics)
- UTF8 String

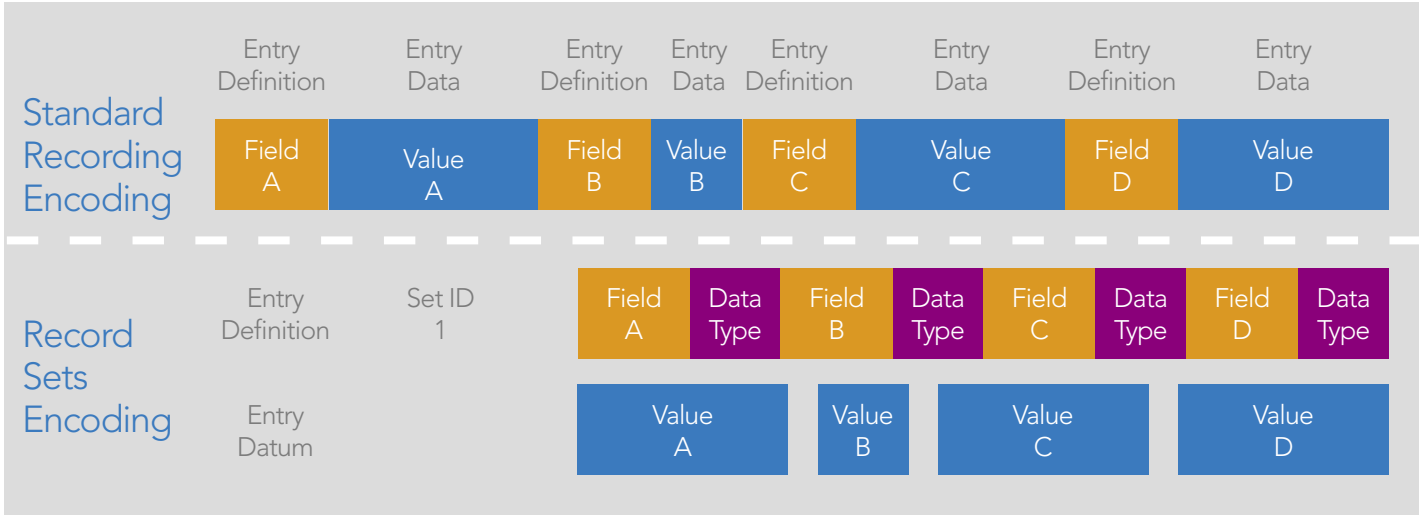
4.2.1.2. Record Sets

OMM provides a "set" concept to optimize bandwidth for record based data. Record based data (i.e. field lists and element lists) is a way of representing logical information as a collection of field/value pairs. The field contains the entry identifier and possibly other attributes while the value contains the actual data content for the entry. Entry identifiers can be self describing (e.g. names and data types in element lists) or reference an independently distributed data dictionary (field identifiers in field lists).

Standard record based data is fully encoded as repeating field data followed by value data. This simple encoding unites the entry data and the entry definition. The key benefits to this encoding are its ease of use and similarity to existing record based formats (e.g. Marketfeed, TibMsg, QForms).

Record sets represent a bandwidth optimization that can be employed to split entry definition from raw entry data for record based content. This allows the entry definitions to be defined once and given an identifier (set id). The actual data for the entries can then be encoded without the definitions; resulting in dramatic bandwidth savings. This optimization can clearly help with repetitively structured records where each record in the structure contains the same entry definitions (e.g. order books, time-series). It can also help with

Figure 14 – Record Set Encodings



repeating record entry definitions across multiple messages (e.g. market price trades, market price quotes, etc.).

Record sets are identifiable and defined at either a **local** or **global scope**.[13] Local scope implies the entry definitions (record sets) are sent/defined in the same message as the entry datum. It is most commonly used for encoding repetitively structured records such as an order book or a time series. Global scope implies the entry definitions (records sets) are sent/defined once, in a record set dictionary, and re-used across many different messages. It is most commonly used for encoding many different messages that contain the same entry definitions (e.g. equity quotes, equity trades, etc.).

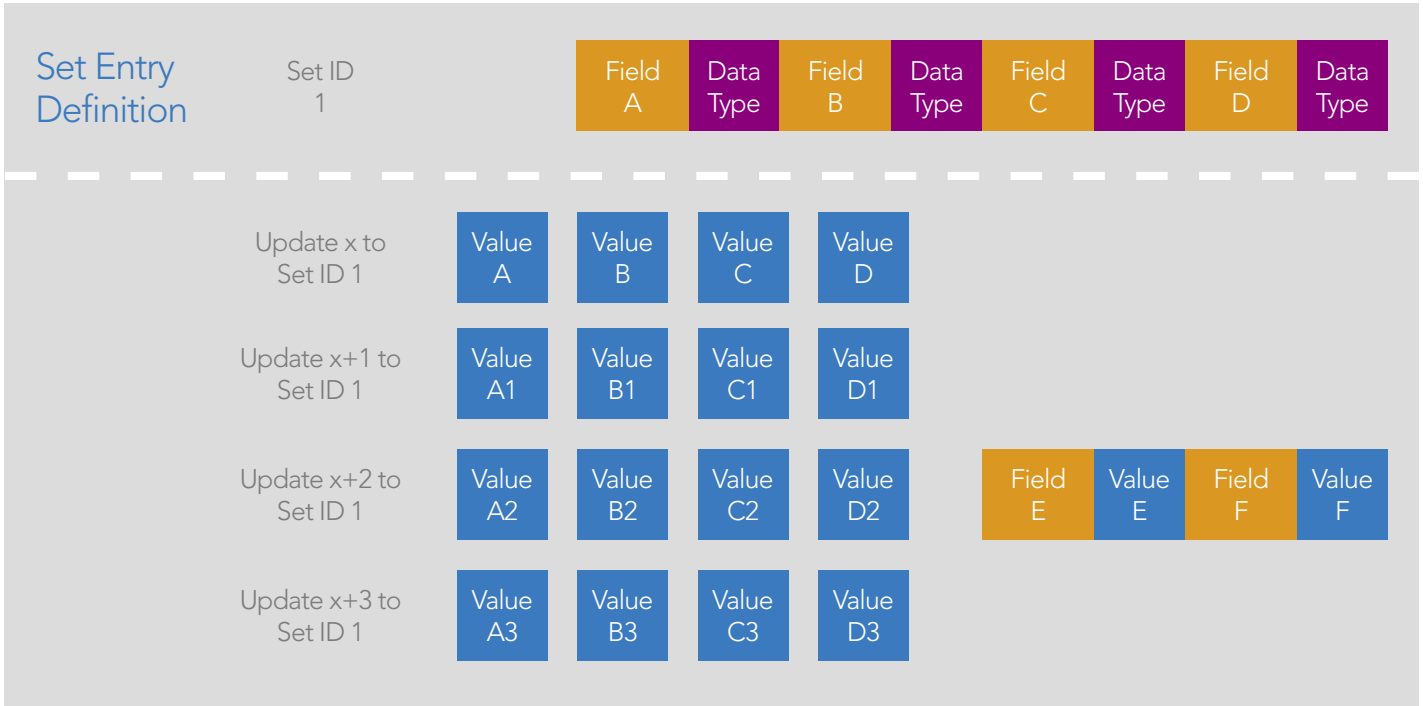
Consumer applications do not need to know the difference between the standard record encoding and the record sets encoding. This is because the decoding libraries default to making content encoded as record sets look like field/value pairs (i.e. standard record encoding). However, providers do need to know the difference since they have to make a choice when encoding the actual record content.

Developers can intermix standard record encoding with record set encoding within a single message. This allows record sets to be defined for the most common cases while supporting the extensibility needed in an open system. This is best described using an example. A system could define a global record set to represent market price quote update. When a quote update is received that also modifies today's high price, the provider could encode the quote data using the normal record set and then append standard record encoding for the extra fields (e.g. fields E and F in Figure 16).

4.2.1.3. Summary Data

Repetitively structured content may have data that pertains to the entire structure. This meta-data typically describes the information contained in each structure entry. For example, summary data could specify the currency of each entries price or rules regarding and sorting of the entries. Structured based data abstractions (e.g. Vector, Map and Series) all support summary data to house this type of information efficiently.

Figure 15 – Extended Record Set



[13] Global scope not supported by RMDS 6.0 or RFA 6.0.

4.2.1.4. Fragmentation

All data abstractions generically support fragmentation across multiple message instances for potentially large sized content. Fragments will be broken on logical entry boundaries in order to simplify the receiving logic. Receiving applications can process each fragment independently (i.e. decode, cache) without waiting for all fragments. Domain message models define whether or not they support fragmentation.

Most structured based data abstractions that support fragmentation provide a total count hint. It is the sending applications suggestion to the total number of entries within the structure across all fragments. The receiving application may choose to use the hint to pre-allocate sufficient memory for caching.

4.2.2. Data Formats

The data abstractions supported through OMM are realized through data formats. This section defines the generic data formats that are available to model all content. The following data formats are defined in terms of structure and the different update semantics needed to keep those structures up-to-date within an event stream.

It is important to understand that these structures are in fact a way of encoding messages and not "in memory" data structures. These data abstractions provide a way that an application can serialize and send a data structure that it uses over the wire.

4.2.2.1. Element List

An element list is the simplest form of logical, or record-based, content. It represents a sequential container of self-describing field/value pair entries; each known as an element entry. Element entries are identified with a string-based tag and self-describe the actual type of data along with the data itself. Element lists do not need any meta-data (i.e. data dictionary) in order to make full sense of the content. An optional element list number, unique within a service, can exist to optimize any caching logic (i.e. element lists with the same element list number should contain the same entries/tags/types).

Element lists can be bandwidth-intensive, but they offer ease-of-use. They can make sense for domains that don't have very high update rates. In certain circumstances, the record set concept (see 4.2.1.2) can be used when encoding an element list in order to reduce the bandwidth needed.

4.2.2.2. Field List

A field list is a more complex, or optimized, form of logical/record-based content. It represents a sequential container of field identifier(or fid)/value pair entries; each known as a field entry. Field entries are identified with a signed 2-byte integer and only contain the actual data for the field. A data dictionary is needed to convert the field identifiers into a tagged name, data type and possible maximum cache length. An optional field list number, unique within a service, can exist to optimize caching logic (i.e. field lists with the same field list number should contain the same entries/fids). Field list numbers are analogous to the record template numbers that are used today.

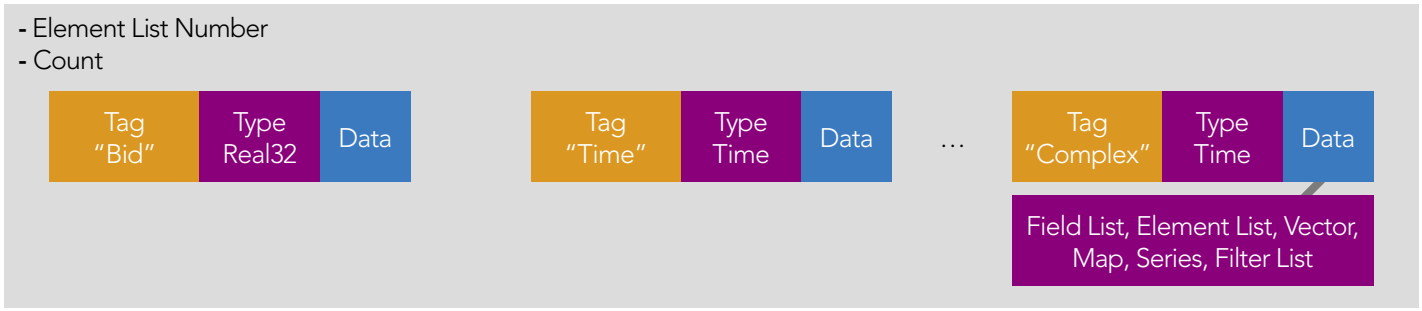


Figure 16 – Element List

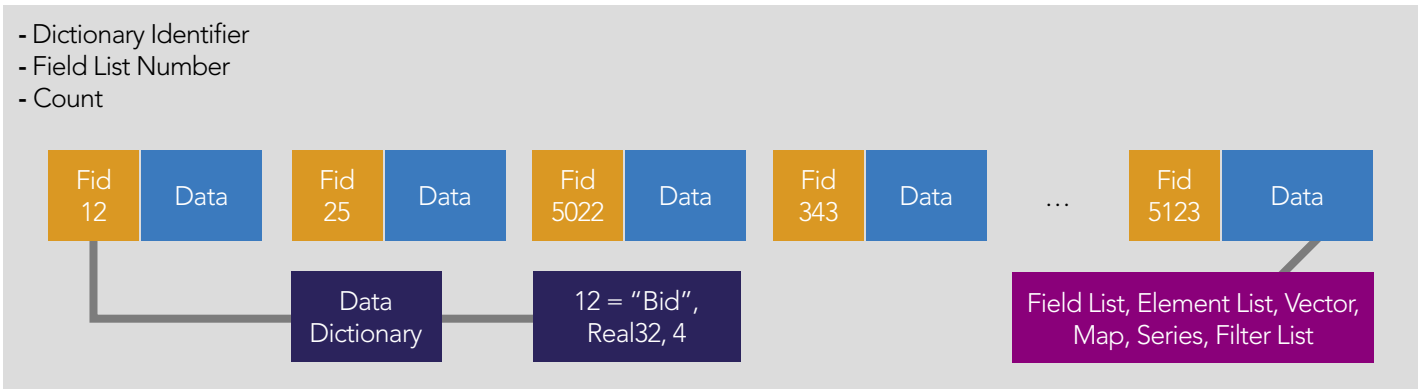


Figure 17 – Field List

Multiple data dictionaries are supported and can be name-spaced when needed. Dictionaries also have versions and can be downloaded from the attached access point. Field lists self-identify the required dictionary needed through the dictionary identifier. A single field list can have fields that reference multiple data dictionaries (e.g. fields from a customer defined dictionary can be added to any vendor provided record).

A field list can be viewed as an element list with a compression technique applied. Instead of passing a full tag/name and data type in every message, an integer-based field identifier is defined. A separate dictionary is needed to convert this fid number into a tag and data type. The record set concept (see 4.2.1.2) can be used when encoding a field list in order to even further reduce the bandwidth needed.

4.2.2.3. Vector

A vector defines a structure that contains highly manipulable position-oriented entries; each known as a vector entry. Each vector entry position is identified by an integer index value. The index starts at 0 and can go as high as a 30-bit unsigned integer value. Entries in the vector can be set, updated or cleared and can optionally each have a separate permissions expression for even finer control. A vector can also optionally support sorting operations (sorted vectors are identified in the response) such as insert and delete.

Vectors provide their largest benefit when combined with other data formats (e.g. vector of field lists, vector of vectors). The data format for each vector entry data is identified once in the vector header (i.e. all vector entries must be the same data format). Vectors optionally contain summary data (see 4.2.1.3) for content that applies to the entire structure. Record set definitions (see 4.2.1.2) can also optionally be defined when vector entries contain repetitive record data.

Vector responses can be fragmented depending on the amount of data in the vector. When this occurs, the encoding application ensures that a vector entry will not be broken across two different fragments. This allows the receiving application to process each fragment independently. An optional total count hint may be provided that indicates the total count for the vector across all fragments in a response.

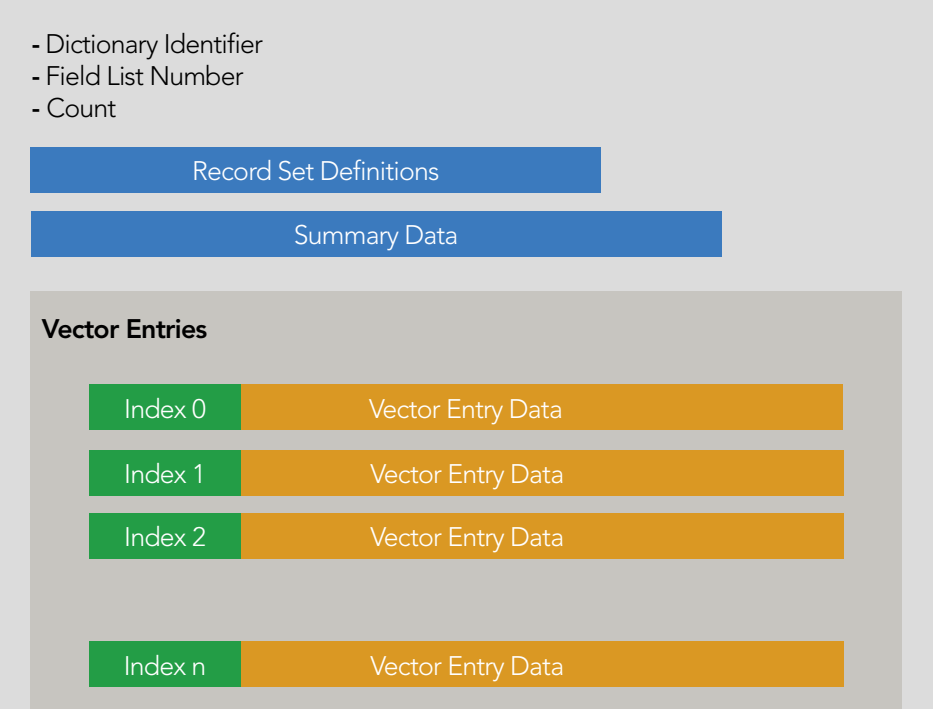


Figure 18 – Vector

4.2.2.4. Map

A map defines a structure of highly manipulable, associative-referenced key-oriented entries; each known as a map entry. The OMM map can be thought of as an STL Map or a generic hash table. Each map entry is identified by a key that can take the form of any basic data type (see 4.2.1.1). Examples could include map entries identified by an ASCII string, binary buffer or even a real number. Entries in the map can be added, updated or deleted and can optionally each have a separate permissions expression for even finer control.

Maps provide their largest benefit when combined with other data formats (e.g. map of field lists, map of vectors). The data format for each map entry data is identified once in the map header (i.e. all map entries must be the same data format). Maps optionally contain summary data (see 4.2.1.3) for content that applies to the entire structure.

Record set definitions (see 4.2.1.2) can also optionally be defined when map entries contain repetitive record data.

Map responses can be fragmented depending on the amount of data in the map. When this occurs the encoding application ensures that a map entry will not be broken across two different fragments. This allows the receiving application to process each fragment independently. An optional total count hint may be provided that indicates the total count for the map across all fragments in a response.

4.2.2.5. Series

A series defines a structure of implicitly-indexed accruable entries; each known as a series entry. A series is typically used to represent repetitively structured data. Series entries cannot be identified and typically have an implicit order (e.g. time, date). Operations on entries are not supported, since there is no way of entry identification.

Series provide their largest benefit when combined with other data formats (e.g. series of field lists, series of vectors). The data format for each series entry data is identified once in the series header (i.e. all series entries must be the same data format). Series optionally contain summary data (see 4.2.1.3) for content that applies to the entire structure. Record set definitions (see 4.2.1.2) can also optionally be defined when series entries contain repetitive record data.

Series responses can be fragmented depending on the amount of data in the series. When this occurs the encoding application ensures that a series entry will not be broken across two different fragments. This allows the receiving application to process each fragment independently. An optional total count hint may be provided that indicates the total count for the series across all fragments in a response.

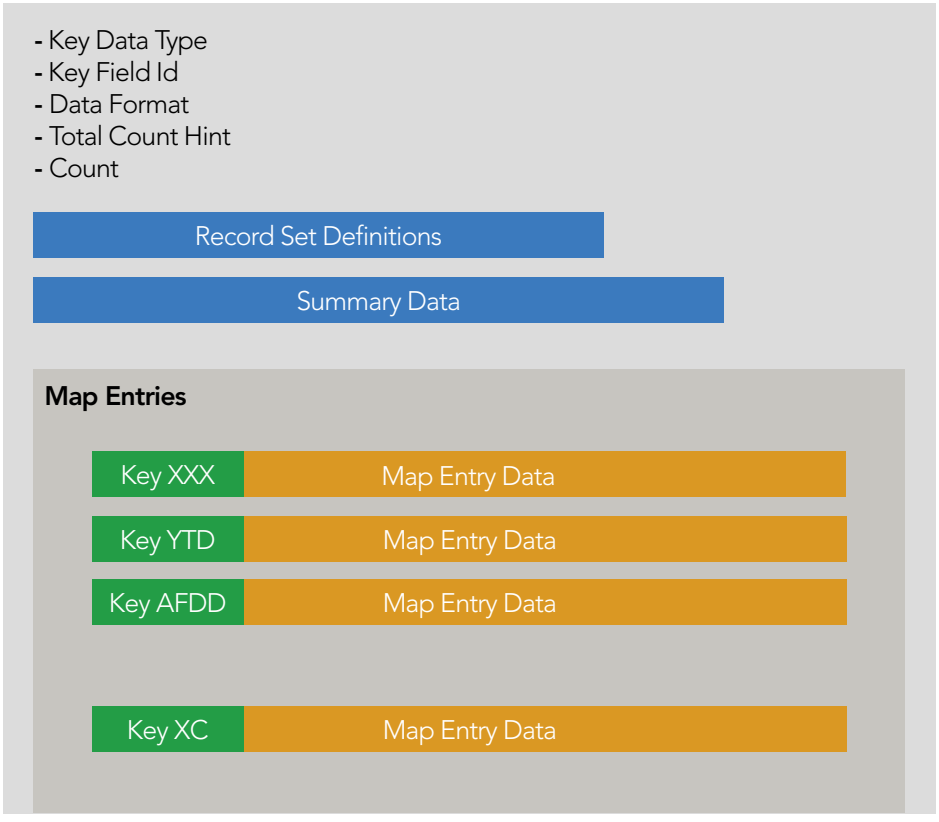


Figure 19 – Map

4.2.2.6. Filter List

Figure 21 – Filter List

A filter list defines a structure of loosely-coupled, associative-referenced entries; each known as a filter entry. Filter lists are defined by the provider and are used to break up information into selectable entries (i.e. they are not meant for field level requests). Filter entries are identified by an 8 bit unsigned integer and can be requested by setting a bit in a bitmap (there are a maximum of 32 filter entries). Entries in the filter list can be set, updated or cleared and can optionally each have a separate permissions expression.

Filter lists provide their largest benefit when combined with other data formats (e.g. filter list of element lists, filter list of maps). Even thought the data format for each filter entry data is identified in the filter list header, filter entries can optionally define different data formats per entry.

Filter list responses can be fragmented depending on the amount of data in the filter list. When this occurs the encoding application ensures that a filter entry will not be broken across two different fragments. This allows the receiving application to process each fragment independently. An optional total count hint may be provided that indicates the total count for the filter list across all fragments in a response.

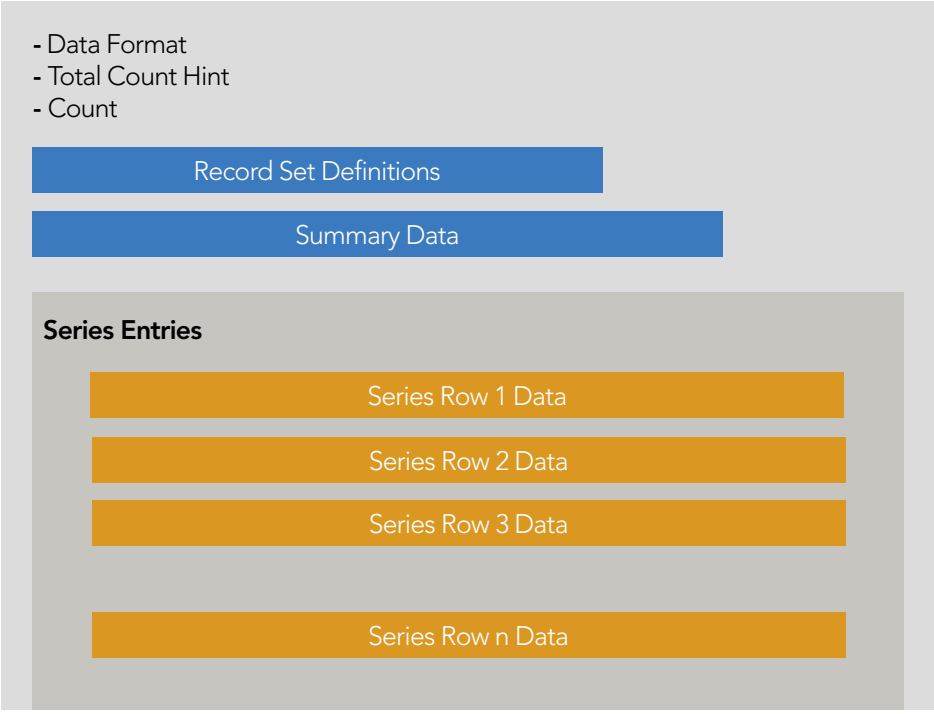


Figure 20 – Series

Figure 21 – Filter List

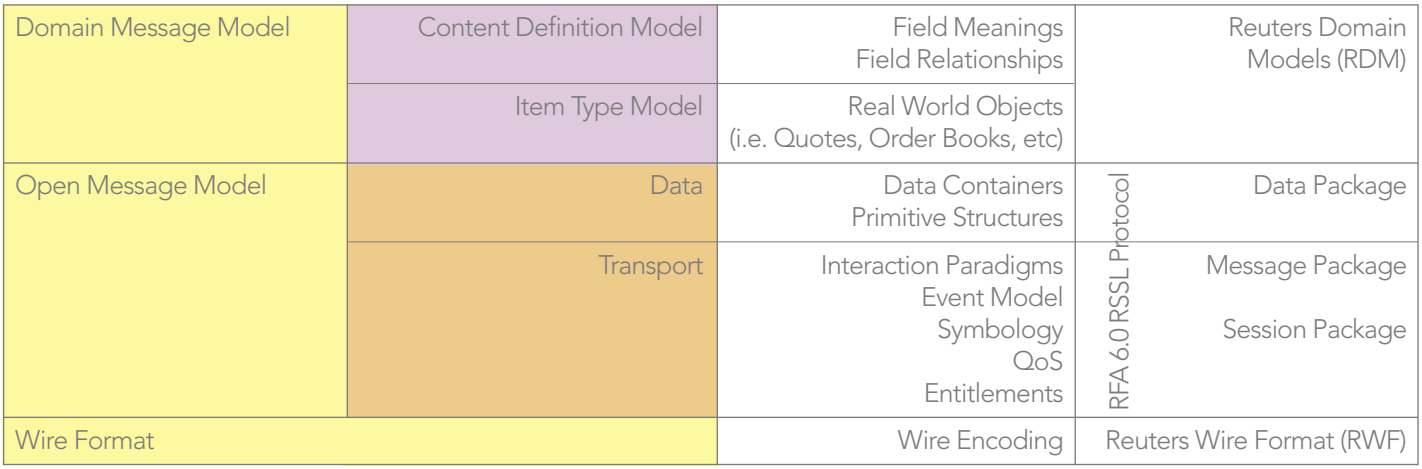
Domain Message Models use the capabilities provided by OMM to define real objects (e.g. Market Price, News Headlines) that are familiar to the industry. Domain Message Model concepts are not understood by RFA or RMDS, thus allowing for new domain models to be created without costly software upgrades. There are a few exceptions within the administrative domain (e.g. Login, Directory) where RFA and RMDS have to perform internal processing.

Developers define Domain Message Models through comprehensive documentation. Message semantics and data representation (e.g. nested data formats) will be fully documented to provide maximum interoperability. Consumer and provider applications must conform to these models in order to properly work together. Domain Message Models can be defined by Reuters or any interested party parties in order to satisfy particular business needs. Where messaging standards already exist (e.g. FIX), Reuters will use these standards modeled on top of OMM will define the Domain Message Models.

The Domain Message Model is divided into two layers, shown in light red above.

- The **Item Type Model** makes up the first layer in the Domain Message Model. It defines the actual object types, their corresponding transport behavior and data representation (i.e. data formats) using OMM abstractions/ concepts. Key attributes (see 4.1.1.2) are chosen and given meaning within the domain. Utilized messages are defined and given full semantic meaning (request/response and any possible event streams). The use of any Quality of Service parameters (see 4.1.1.4) is defined and given concrete meaning. Any attributes that make up the extended header will also be defined.
- The **Content Definition Model** builds upon the Item Type Model in order to complete the domain message model. This important and often not completely specified layer defines any field meanings and relationships on top of the Item Type Models. A single Item Type Model can have many different Content Definition Models (e.g. different field identifiers, different symbology). Content Definition Models can include data dictionaries, enumerations information and any required/optional field definitions. Some Item Type Models (e.g. transactions) require a stricter Content Definition Model than currently defined for price discovery.

Figure 22 – Reuters Data Model Architecture



5. Domain Message Models

5.1. Reuters Domain Models

Reuters has defined a set of domain models that will be used by all Reuters providers and consumers. Any applications that want to achieve maximum interoperability with Reuters providers and consumers should utilize these models. The definitions are broken into Item Type Models and Content Definition Models. Therefore, applications can still interoperate as long as they conform to the Item Type Models.

The currently defined administrative Reuters Domain Models are:

- **Login** – login a user to a system access point (see 2.3) and create a context for the user within the system.
- **Directory** – directory and detailed information about service providers available to consumers.
- **Dictionary** – provides access to all required dictionaries (e.g. data dictionaries, enumerations files, etc.).

The currently defined Reuters Domain Models for instrument-based market data are:

- **Market Price** – updating trades, quotes and inside top of book quotes (i.e. level I content).
- **Market-by-Order** – updating instrument market information sorted by order (i.e. full order book – level II content).
- **Market-by-Price** – updating top of book instrument market information sorted by best price (i.e. market depth – level II content).
- **Market Maker** – updating market maker quotes and trade information from exchanges.
- **Symbol List** – updating lists of symbols/instruments (e.g. .AV.O, S&P500, NASDAQ, etc.).

The previous list is not exhaustive as new domains are being defined. Check the Reuters Customer Zone for the latest Domain Models.

The sections that follow contain the definitions of a few of the Reuters Item Type Models. They are useful examples of how OMM is used to create Domain Message Models within Reuters.

5.1.1. Login

The login item type is used to create a context within an access point for all other types of interactions. It is the special item type model that has to be used in order to use the system. Logins are the first request that needs to be done and also have to be streaming to maintain the user context within the access point. Access points have special logic to handle logins and utilize them to retrieve permissions information for the user. This permissions profile will be used to authorize all of the other item type model interactions.

Figure 23 – Login

Message Classes

- Request
- Refresh
- Update
- Status
- Close
- Ack

Key – User Identity

- Name = User Identity
- Name Type = User Identity Type

User IdentityTypes

- User Name
- Email Address

Request Data – Element List

Reply Data – Element List

The transport semantics used with Login are:

Interaction Paradigm
- Request/Response with Interest

Key – User Identity
- Name – the actual user identity
- Name Type – the type of user identity contained within the Name (e.g. username, email address, etc.)

Unused Capabilities
- Does not use priority
- Does not use quality of service
- Does not use event stream groups

Request Message
- Request a login into an access point
- Payload data contains login options/ parameters

Refresh Message
- Response to Request or unsolicited Refresh to reset all login context data
- Stream State Open implies login success
- Stream State Closed implies login denied
- Data in response contains login profile information
- Data contained in single refresh message
- Refresh Key could define another way of identifying the same user

Update Message
- Update to some login context data
- Cannot be conflated

Status Message
- Status change to logged-in context
- Stream State Closed implies forced logoff

Close Message
- Logoff an already logged-in context or close a pending login request

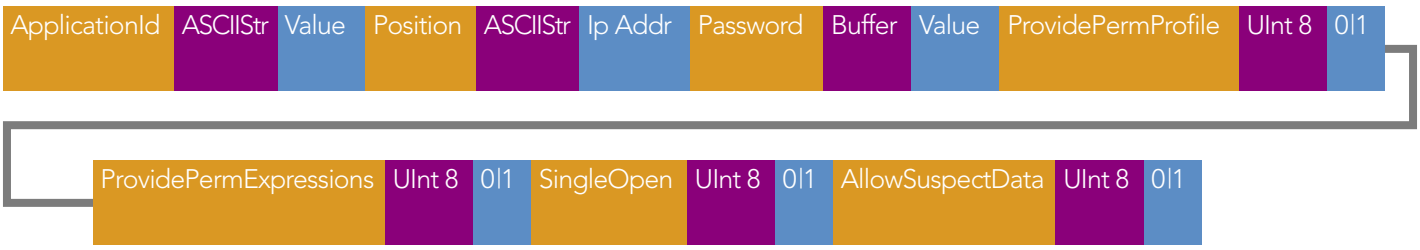
Ack Message
- Optionally used to acknowledge a close (logoff)

This streaming model supports data in the request that is used to pass login parameters to the access point. The response message state indicates the success or failure of the login request. Upon success, the response data may contain login specific information for the session that was either requested or sent from the access point. Updates are used to modify some of the content provided in the response data that might change over the life of a session (e.g. a user's permissioning profile).

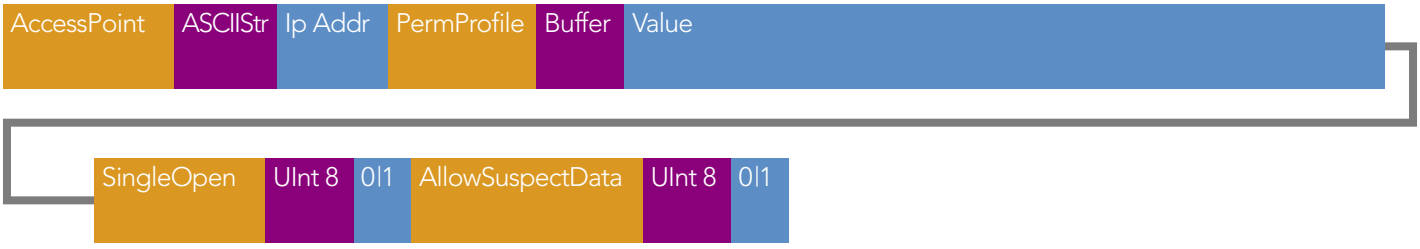
The data formats used during login requests and replies are depicted in Figure 24. Elements lists are used for both request data and reply data due to their simplicity, flexibility, and the fact the login messages are not very frequent. The parameters that need to be sent with the login request are contained within an element list in the request data. Any information, or parameters, that need to be sent back to the user are contained within an element list in the refresh data.

Figure 24 – Login Data

Request Data – Element List



Reply Data – Element List



Even though the login item type model has special handling, the generic RMDS last-value cache can optionally be used to cache login data.

Figure 25 gives an example of some of the request, and reply, login data that may be present. These lists of parameters may be extended in the future as new capabilities are added. The permissions profile for a user can optionally be retrieved and will exist within the "PermProfile" element of the reply data.

5.1.1.1. Data Encodings

Standard element list encoding is used in both the request and reply data. Separate "NoRefresh" requests can be performed to modify parameters of the access point. Unsolicited refresh messages may be received and are used to reset all of the reply data (e.g. new user profile). Updates are used to update parts of the data sent in the refresh messages (e.g. modification to a user's profile).

5.1.1.2. Example

Figure 25 shows an example scenario of the login item type model.

A login request is sent to the access point; it contains the user identity key, flags to indicate streaming, and an element list that contains any parameters. The access point asynchronously responds with a login success; it is represented as a single

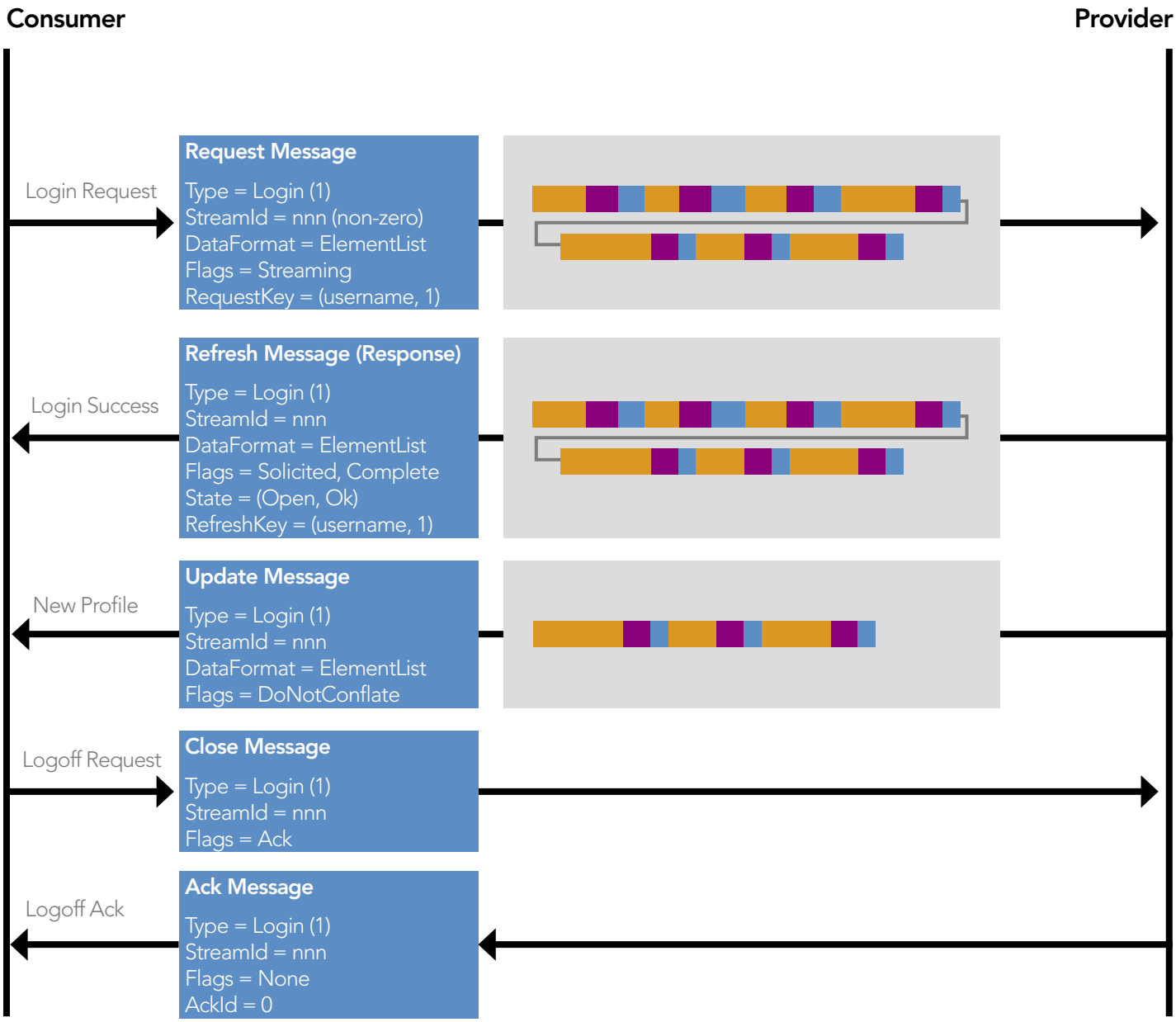


Figure 25 – Login Example

solicited refresh message that contains an open stream state, an ok data state and an element list containing any information requested (e.g. permissions profile) or any default parameters.

At this point the consumer can use any of the other item type models with the connected access point. The access point can, at any time, send an update message that contains modifications to any requested (e.g. permissions profile) or default data sent in the refresh.

When the user wants to logoff the system, a close message is sent for the stream identifier used in the original login request. Since the close contained the optional ack flag, the user will receive an acknowledgment of the close from the access point.

5.1.2. Market Price

The term "Market Price" is used to denote information which contains trades, indicative quotes and the inside top of book quotes. It includes the last traded price(s), best bid(s)/offer(s), related value data such as: Names, Codes, etc. and the related derived data such as: Net Change, pen, Close, High(s), Low(s), etc. The current Reuters model for "Level-1" data forms the basis of the Market Price domain. It includes different asset classis including equities, fixed income, commodities, money, FX and contributed quote data.

Chains, IDN encoded time-series (TS1), IDN encoded time & sales, and IDN record pages being published by existing MarketFeed providers would also be represented by the Market Price domain.

The transport semantics used with Market Price are:

Interaction Paradigm
- Request/Response with or without (snapshot) Interest

Key – Instrument Key
- Service ID – the identifier of the service for the request
- Name – the symbol of the instrument
- Name Type – the symbology of the symbol (e.g. RIC, ISIN, etc.)

Options
- Supports priority
- Quality of service applies
- Event stream groups apply
- Sequence number contains sequence number from exchange

Request Message
- Request an instruments market price information from an access point (either streaming or snapshot)

Refresh Message

Response to Request or unsolicited Refresh to reset all market price/event stream data
- Data in response contains all of the market price information for the requested instrument
- Data contained in single refresh message (single response)
- Refresh Key could define the way the actual service identifies the instrument (ISIN as opposed to a RIC)

Update Message
- Update to the fields, that where received in the refresh, that have changed value
- Updates can be conflated (last field values sent)

Status Message
- Status change to event stream

Close Message
- Close an already open event stream or close a pending request

Ack Message
- Optionally used to acknowledge a close

This optional event streaming model provides a full image in a single refresh/response message. Updates are used to modify the fields of the image data that have changed based on some market event (e.g. quote, trade, news event, etc.). Market Price information is one of multiple types of data available for market data instruments.

Message Classes
- Request
- Refresh
- Update
- Status
- Close
- Ack

Key – Instrument Key
- Service Id
- Name = Symbol
- Name Type - Symbology

Instrument Key Types
- RIC
- Street Symbol
- ISIN
- CUSIP

Market Price Data – Field List
- Dictionary Id
- Field List Number
- Count

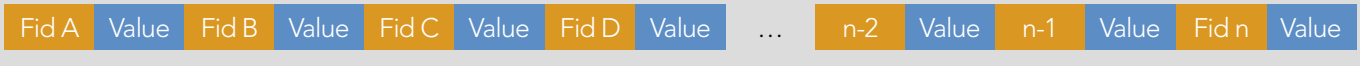
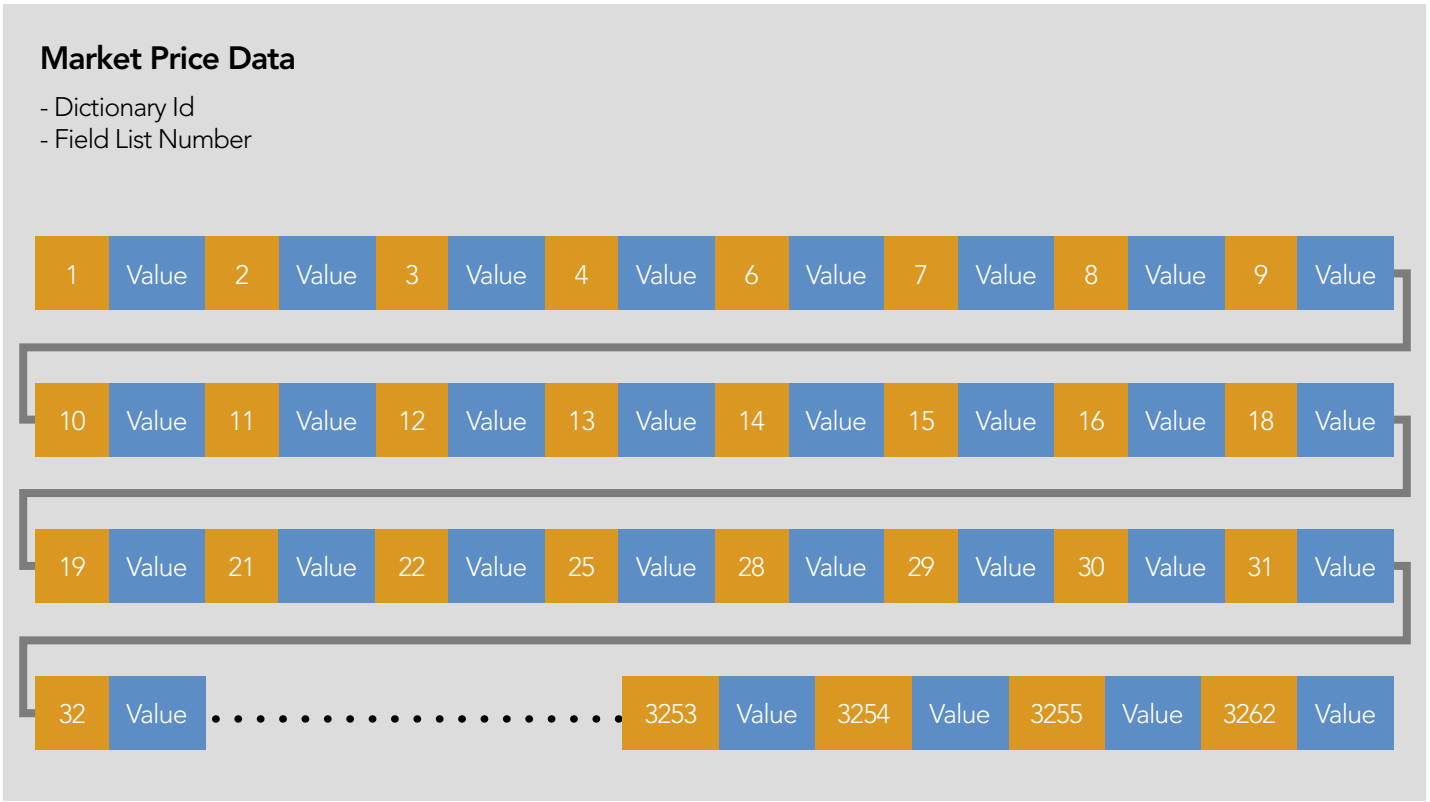


Figure 26 – Market Price

The format used to represent Market Price data is depicted in Figure 26. The field list data format is used to represent field/value pairs in a bandwidth-efficient manner. This is important, since market price information can potentially update at very high rates. All of the data that makes up market price information for an instrument is contained within individual fields in the field list. The collection of these fields, or the field list itself, then represents all of the market price information. A dictionary identifier will always be present and an optional field list number may exist for efficient caching. The standard RMDS last-value cache can optionally be used to cache Market Price data.

The content or list of fields that make up a particular type of market price information is determined by the underlying type of instrument. For example, a field list representing an equity instrument will not contain the same fields as one that represents the spot market for a currency. Figure 27 gives an example of the resulting fields that make up a US equity.

Figure 27 – Market Price Data



5.1.2.1. Data Encodings

Figure 28 shows the data format layout for the different message types within the Market Price domain. The refresh/response holds an image that contains all of the fields/values that make up the instrument. It also defines the dictionary identifier and optional field list number. Market events (e.g. Trades/Quotes) for the instrument are represented as updates and contain only the fields/values that are modified by the event. Standard field list encoding is used in both the refresh and update messages for this domain. Record sets, in conjunction with global field list set definitions, may optionally be used in the future to even further reduce message sizes.

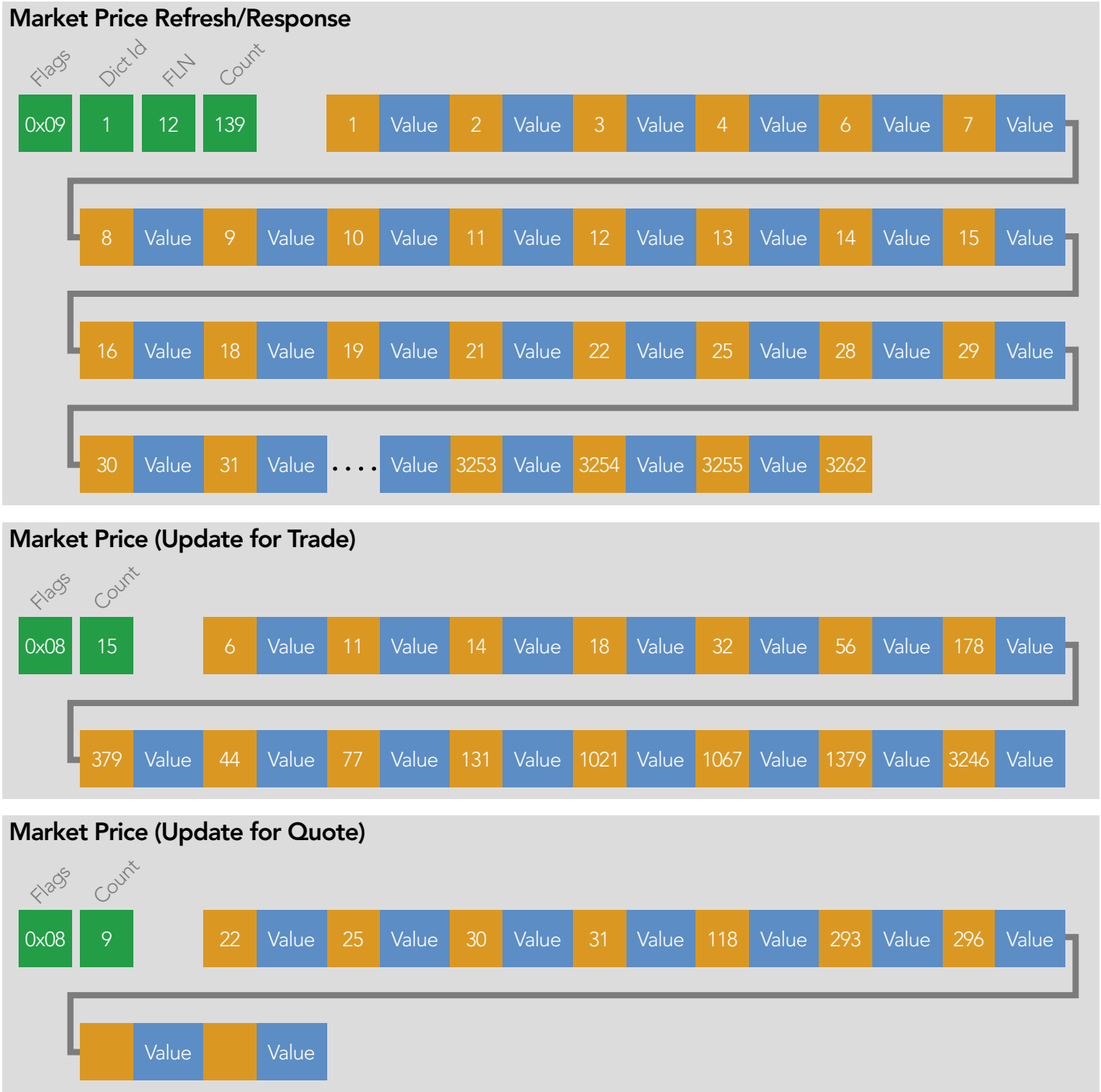


Figure 28 – Market Price Data Encodings

5.1.2.2. Example

Figure 29 shows an example scenario for the market price item type model.

A request is sent to the access point; it contains the instrument key (service id, symbol and symbology) and flags to indicate streaming. The access point asynchronously responds with a market price image; it is represented as a single solicited refresh message that contains an open stream state, an ok data state and a field list containing the data for the instrument. The refresh also defines the group identifier, optional data source sequence number and event stream quality of service (e.g. Realtime/ TickByTick).

At this point the event stream is open for the instrument and the consumer can asynchronously receive updates for the instrument data. The update message can optionally contain an update type (e.g. quote, trade) and a data source (e.g. exchange) sequence number. The sequence number does not always increment by one, since each exchange generates the number differently.

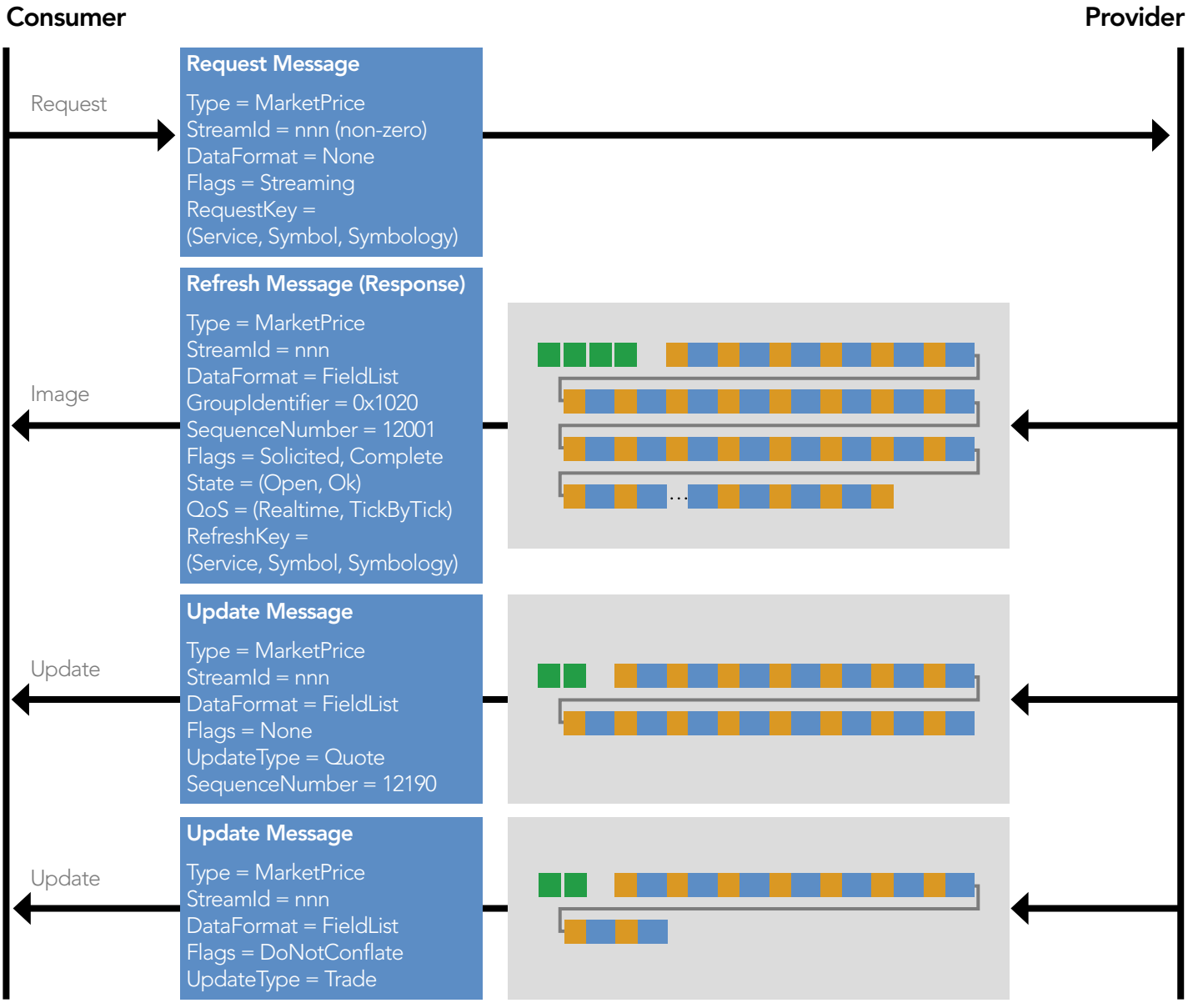


Figure 29 – Market Price Example

5.1.3. Market by Order

The term "Market by Order" is used to denote information which contains full instrument order book content. It includes all of the orders for the security, each containing fields like price, size, side, time and other related information. New orders can be added to the book and existing orders can either be modified (e.g. partial fill) or deleted from the book.

The transport semantics used with Market by Order are:

Interaction Paradigm
- Request/Response with or without (snapshot) Interest

Key – Instrument Key
- Service ID – the identifier of the service for the request
- Name – the symbol of the instrument
- Name Type – the symbology of the symbol (e.g. RIC, ISIN, etc.)

Options
- Supports priority
- Event stream groups apply
- Sequence number contains sequence number from exchange

Request Message
- Request an instruments order book information from an access point (either streaming or snapshot)

Refresh Message
- Response to Request or unsolicited Refresh to reset all order book event stream data
- Data in response contains all information for the orders within that response.
- Full image can be broken across multiple refreshes/responses due to the potential size of the book.
- Refresh Key could define the way the actual service identifies the instrument (ISIN as opposed to a RIC)

Update Message
- Update to the orders that were received in the refresh (e.g. add, delete, update)
- When updating an order only contain the fields that have changed in the order

Status Message
- Status change to event stream

Close Message
- Close an already open event stream or close a pending request

Ack Message
- Optionally used to acknowledge a close

Message Classes
- Request
- Refresh
- Update
- Status
- Close
- Ack

Key – Instrument Key
- Service Id
- Name = Symbol
- Name Type - Symbology

Instrument Key Types
- RIC
- Street Symbol
- ISIN
- CUSIP

Order Book Data – Map of Field List
- Key Data Type – Buffer
- Data Format – Field List
- Count

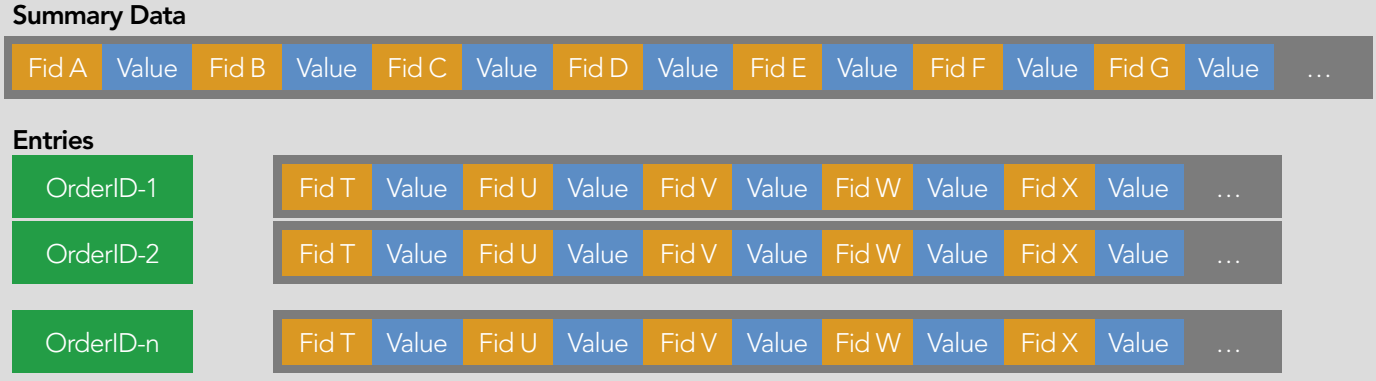


Figure 30 – Market by Order

Order books can be accessed through snapshots or, if needed, an asynchronous event stream can also be created. The full order book data comes back as an image that may span multiple refresh/response messages. Updates are used to modify the order book based on a market activity (e.g. add order, modify order, delete order). Market by Order information is one of multiple types of data available for a market data instrument.

The format used to represent Market by Order data is depicted in Figure 30. A map of field lists is used to represent the two-dimensional structure that makes up the book. Orders are represented by map entries that contain field/value pairs for the information that makes up that order. The map entries are identified by the order identifier as received by the data source (e.g. exchange) and always contain the same set of fields. The map will also contain summary data that is made up of field/value pairs for information that is pertinent to the entire book (e.g. currency, exchange state, etc.). The dictionary identifier in the summary data defines the dictionary to be used for the entire book. Standard RMDS last-value caching can optionally be used to cache Market by Order data.

Figure 31 gives an example of the resulting data structure for an Order Book. Summary data is used to provide fields that apply to the entire book. Some of the fields that might exist within summary data include PE, Currency, Exchange Identifier, Trade Units, etc. The actual orders are separated by the order identifiers and contain the actual fields that make up the order. Orders typically contain fields such as Price, Side, Size, Identifier and Time. The fields within summary data and the fields within the orders can easily be extended to provide exchange-specific or other specific content.

Figure 31 – Market by Order Data

5.1.3.1. Data Encodings

Figure 32 shows the data format layout of the image within the refresh/response messages for the Market by Order domain. It uses the record set optimization since the book contains many entries with the same fields. The layout of the record set for each entry is given within the list set definitions. Summary data is then defined using standard field list encoding rules. The field list within the summary data also defines the dictionary identifier for all of the fields within the book and an optional field list number for the layout of the fields that make up the summary data. Order book entries, identified by the order id, are then defined (Added), each containing the field data for the order. Record set encoding rules are used for the field lists within entries in order to save bandwidth. Also note that the set identifier for the field list set data in each entry defaults to 0, since there is no specific set ID defined.

Figure 32 – Market by Order Image Encodings

The encoding rules used for data within update messages are shown in Figure 33. When fields are present (i.e. Add, Update) standard field list encoding rules are used for simplicity. An update can add an order, update fields within an existing order or delete an existing order. Updates typically act upon a single entry due to the nature of an order book. A single update can act upon multiple entries when some form of item-level update batching is employed.

5.1.3.2. Example

The example scenario for the market by order item type model is shown in Figure 34.

A request is sent to the access point; it contains the instrument key (service id, symbol and symbology) and flags to indicate streaming. The access point asynchronously responds with the first part of an order book image; it is represented as a solicited refresh message that contains an open stream state, an ok data state and a map containing the data for the instrument. The first refresh also contains the group identifier and optional data source sequence number.

Before the image is fully sent, the provider can send an update to the event stream representing the order book. In fact this update may be for data that has not yet been sent to the application. The provider then sends the final part of the image within another refresh message that is marked as solicited and Complete.

At this point the event stream is open for the order book and the consumer can asynchronously receive updates for the data (Add, Delete, Update). The update message can optionally contain a data source sequence number that does not always increment by one.

Figure 33 – Market by Order Image Encodings

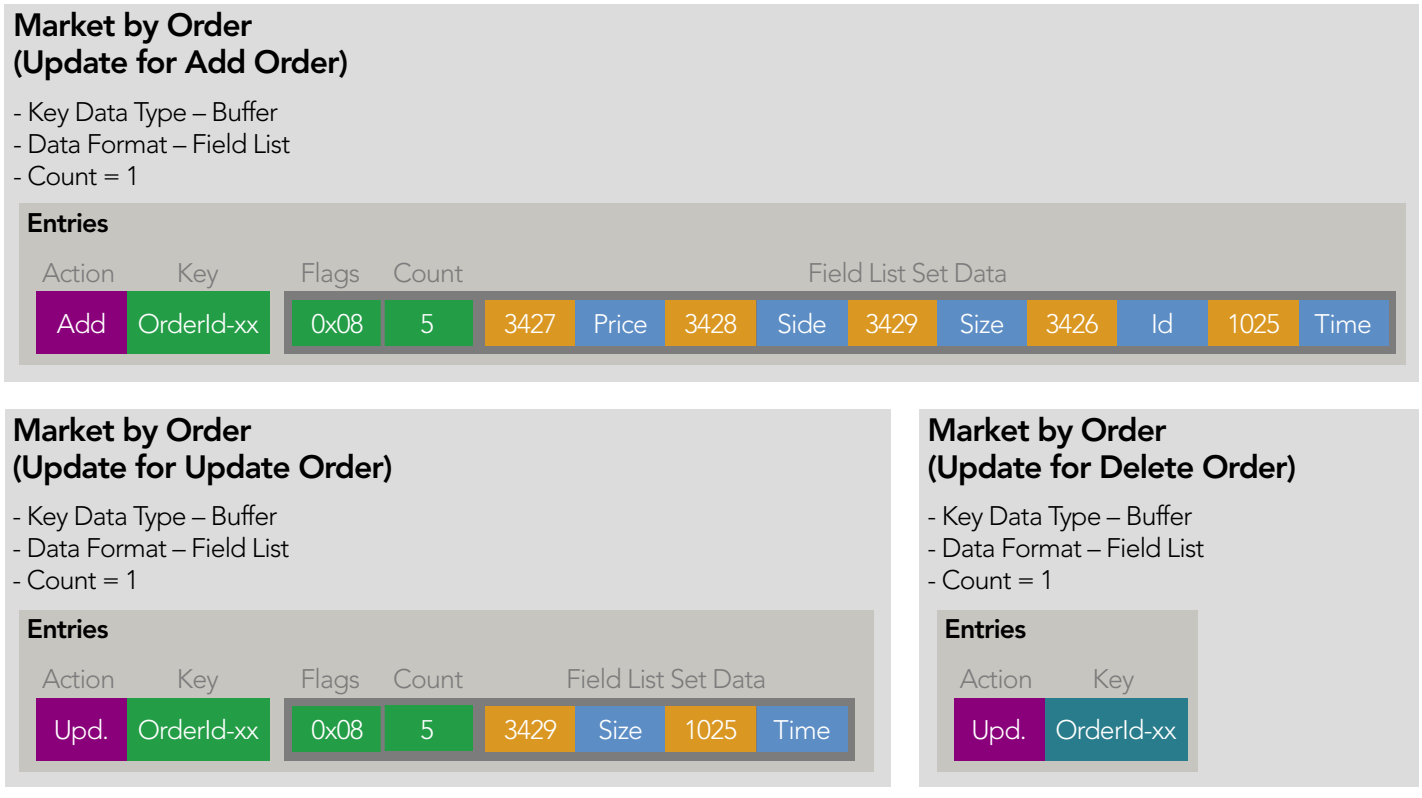
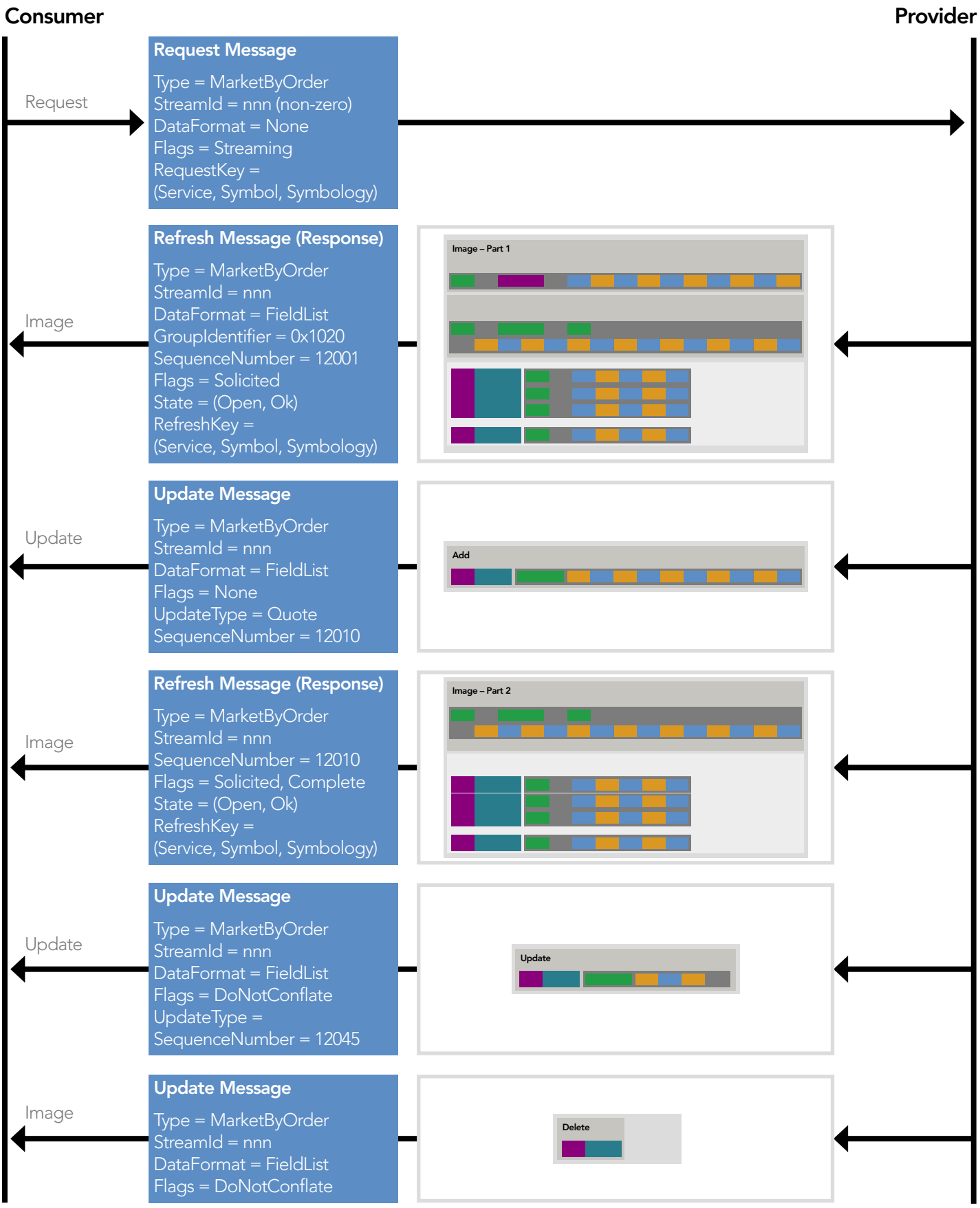


Figure 34 – Market by Order Example



5.2. Defining new types

This is where OMM becomes really very interesting. Because OMM is truly open, you may extend an existing type or create your own data type. You could define a yield curve, a volatility surface, a complex weather derivative contract, or a multi-asset portfolio as a few examples. You can define the types that your application needs.

New types are defined by specifying the Item Type Model and the Content Definition Model. To create a new type:

- Define the transport semantics including the interaction paradigm, the request key, the supported options, the request message, the refresh message, and other messages like the update, status, close and ACK.
- Design the data encoding using the OMM data abstractions. Define what, if any, optimizations are used including Summary Data and Record Sets. An example of this is in Figures 31 – 33.
- Define the interaction scenario as in Figure 34.
- Define the Content Definition Model, including fields and data dictionaries dpecific to your provider's content.
- Publish and share documentation for developers in your organization or to developers that will interact with your provider of this content. Developers that need to consume this type will write RFA code to handle this new type model.

If you find that you need assistance as you develop your first few types, Reuters will be happy to help customers with the design needs for OMM domain models via training, consulting, or support. In some cases an architect in the development organization may want to work directly with you if this is a type that will have a wide applicability. Please contact Reuters through your account team to discuss the best way of gaining assistance. As an RDC member, you may also want to interact with other developers in our RDC Developer Forums to discuss creating a new type.

Published by Reuters Limited, The Reuters Building, South Colonnade, Canary Wharf, London, E14 5EP.
IM Jun 06 0519

REUTERS