

Enterprise Message API Java Edition 3.6.2.G2

ENTERPRISE MESSAGE API CONFIGURATION GUIDE

Document Version: 3.6.2.G2
Date of issue: August 2021
Document ID: EMAJ362CG.210



© **Refinitiv 2016 - 2021**. All rights reserved.

Republication or redistribution of Refinitiv content, including by framing or similar means, is prohibited without the prior written consent of Refinitiv. 'Refinitiv' and the Refinitiv logo are registered trademarks and trademarks of Refinitiv.

Any software, including but not limited to: the code, screen, structure, sequence, and organization thereof, and its documentation are protected by national copyright laws and international treaty provisions. This manual is subject to U.S. and other national export regulations.

Refinitiv, by publishing this document, does not guarantee that any information contained herein is and will remain accurate or that use of the information will ensure correct and faultless operation of the relevant service or equipment. Refinitiv, its agents, and its employees, shall not be held liable to or through any user for any loss or damage whatsoever resulting from reliance on the information contained herein.

Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 1 |
| 1.1 | About this Manual | 1 |
| 1.2 | Audience | 1 |
| 1.3 | Acronyms and Abbreviations | 1 |
| 1.4 | References | 2 |
| 1.5 | Documentation Feedback | 2 |
| 1.6 | Document Conventions | 3 |
| 1.6.1 | <i>Typographic</i> | 3 |
| 1.6.2 | <i>Field and Text Values</i> | 3 |
| 1.6.3 | <i>Boolean Values</i> | 3 |
| 2 | EMA Configuration General Overview | 4 |
| 2.1 | About Message API Configuration | 4 |
| 2.2 | Parameter Overview | 4 |
| 2.3 | Default Behaviors | 5 |
| 3 | Configuration Groups | 6 |
| 3.1 | ConsumerGroup | 6 |
| 3.1.1 | <i>Generic XML Schema for ConsumerGroup</i> | 6 |
| 3.1.2 | <i>Setting a Default Consumer</i> | 6 |
| 3.1.3 | <i>Configuring Consumers in a ConsumerGroup</i> | 7 |
| 3.1.4 | <i>Consumer Entry Parameters</i> | 7 |
| 3.2 | Provider Groups | 11 |
| 3.2.1 | <i>Generic XML Schema for Provider Group</i> | 11 |
| 3.2.2 | <i>Setting a Default Provider</i> | 11 |
| 3.2.3 | <i>Configuring a Provider in a ProviderGroup</i> | 12 |
| 3.2.4 | <i>Provider Entry Parameters</i> | 12 |
| 3.3 | Channel Group | 17 |
| 3.3.1 | <i>Generic XML Schema for ChannelGroup</i> | 17 |
| 3.3.2 | <i>Universal Channel Entry Parameters</i> | 18 |
| 3.3.3 | <i>Parameters for Use with Channel Type: RSSL_SOCKET</i> | 20 |
| 3.3.4 | <i>Parameters for Use with Channel Types: RSSL_HTTP</i> | 21 |
| 3.3.5 | <i>Parameters for Use with Channel Types: WEBSOCKET</i> | 22 |
| 3.3.6 | <i>Parameters for Use with Channel Types: RSSL_ENCRYPTED</i> | 22 |
| 3.3.7 | <i>Example XML Schema for Configuring ChannelSet</i> | 23 |
| 3.3.8 | <i>Example ChannelSet XML Configuraion</i> | 23 |
| 3.3.9 | <i>Example Programmatic Configuration for ChannelSet</i> | 24 |
| 3.4 | Server Group | 26 |
| 3.4.1 | <i>Generic XML Schema for ServerGroup</i> | 26 |
| 3.4.2 | <i>Server Entry Parameters</i> | 27 |
| 3.4.3 | <i>Parameters for Use with ServerType WEBSOCKET</i> | 28 |
| 3.5 | Dictionary Group | 29 |
| 3.5.1 | <i>Generic XML Schema for DictionaryGroup</i> | 29 |
| 3.5.2 | <i>Dictionary Entry Parameters</i> | 29 |
| 3.6 | Directory Group | 30 |
| 3.6.1 | <i>Generic XML Schema for Directory Entry</i> | 30 |
| 3.6.2 | <i>Setting Default Directory</i> | 30 |
| 3.6.3 | <i>Configuring a Directory in a DirectoryGroup</i> | 31 |
| 3.6.4 | <i>Service Entry Parameters</i> | 31 |
| 3.6.5 | <i>InfoFilter Entry Parameters</i> | 31 |
| 3.6.6 | <i>StateFilter Entry Parameters</i> | 35 |

| | | |
|----------|---|-----------|
| 3.6.7 | <i>Status Entry Parameters</i> | 35 |
| 3.7 | Global Configuration | 36 |
| 3.7.1 | <i>Generic XML Schema for GlobalConfig</i> | 36 |
| 3.7.2 | <i>Parameters</i> | 36 |
| 4 | EMA Configuration Processing | 37 |
| 4.1 | Overview and Configuration Precedence..... | 37 |
| 4.2 | Default Configuration | 37 |
| 4.2.1 | <i>Default Consumer Configuration</i> | 37 |
| 4.2.2 | <i>Default Provider Configurations</i> | 38 |
| 4.3 | Processing EMA's XML Configuration File | 39 |
| 4.3.1 | <i>Reading the Configuration File</i> | 39 |
| 4.3.2 | <i>Use of the Correct Order in the XML Schema</i> | 40 |
| 4.3.3 | <i>Processing the Consumer "Name"</i> | 41 |
| 4.3.4 | <i>Processing the Provider "Name"</i> | 41 |
| 4.4 | Configuring EMA Using Function Calls | 42 |
| 4.4.1 | <i>EMA Configuration Method Calls</i> | 42 |
| 4.4.2 | <i>Using the host() Function: How Host and Port Parameters are Processed</i> | 44 |
| 4.5 | Programmatic Configuration | 45 |
| 4.5.1 | <i>OMM Data Structure</i> | 45 |
| 4.5.2 | <i>Creating a Programmatic Configuration for a Consumer</i> | 46 |
| 4.5.3 | <i>Example: Programmatic Configuration of the Consumer</i> | 47 |
| 4.5.4 | <i>Creating a Programmatic Configuration for a Provider</i> | 49 |
| 4.5.5 | <i>Example: Programmatic Configuration of a Provider</i> | 50 |

1 Introduction

1.1 About this Manual

This document is authored by Enterprise Message API architects and programmers. Several of its authors have designed, developed, and maintained the Enterprise Message API product and other Refinitiv products which leverage it. As such, this document is concise and addresses realistic scenarios and use cases.

This guide documents the functionality and capabilities of the Enterprise Message API Java Edition . The Enterprise Message API can also connect to and leverage many different Refinitiv and customer components. If you want the Enterprise Message API to interact with other components, consult that specific component's documentation to determine the best way to configure for optimal interaction..

This document explains the configuration parameters for the Enterprise Messaging API (simply called the Message API). Message API configuration is specified first via compiled-in configuration values, then via an optional user-provided XML configuration file, and finally via programmatic changes introduced via the software.

Configuration works in the same fashion across all platforms.

1.2 Audience

This manual provides information that aids software developers and local site administrators in understanding Enterprise Message API configuration parameters. You can obtain further information from the *Enterprise Message Java Edition API Developer's Guide*.

1.3 Acronyms and Abbreviations

| ACRONYM / TERM | MEANING |
|--------------------------------|---|
| ADH | Refinitiv Real-Time Advanced Data Hub is the horizontally scalable service component within the Refinitiv Real-Time Distribution System providing high availability for publication and contribution messaging, subscription management with optional persistence, conflation and delay capabilities. |
| ADS | Refinitiv Real-Time Advanced Distribution Server is the horizontally scalable distribution component within the Refinitiv Real-Time Distribution System providing highly available services for tailored streaming and snapshot data, publication and contribution messaging with optional persistence, conflation and delay capabilities. |
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Interchange |
| Enterprise Message API | The Enterprise Message API (EMA) is an ease of use, open source, Open Message Model API. EMA is designed to provide clients rapid development of applications, minimizing lines of code and providing a broad range of flexibility. It provides flexible configuration with default values to simplify use and deployment. EMA is written on top of the Enterprise Transport API (ETA) utilizing the Value Added Reactor and Watchlist features of ETA. |
| Enterprise Transport API (ETA) | Enterprise Transport API is a high performance, low latency, foundation of the Refinitiv Real-Time SDK. It consists of transport, buffer management, compression, fragmentation and packing over each transport and encoders and decoders that implement the Open Message Model. Applications written to this layer achieve the highest throughput, lowest latency, low memory utilization, and low CPU utilization using a binary Refinitiv Wire Format when publishing or consuming content to/from Refinitiv Real-Time Distribution Systems. |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol (Secure) |

Table 1: Acronyms and Abbreviations

| ACRONYM / TERM | MEANING |
|---|---|
| OMM | Open Message Model |
| QoS | Quality of Service |
| RDM | Refinitiv Domain Model |
| Refinitiv Real-Time Distribution System | Refinitiv Real-Time Distribution System is Refinitiv's financial market data distribution platform. It consists of the Refinitiv Real-Time Advanced Distribution Server and Refinitiv Real-Time Advanced Data Hub. Applications written to the Refinitiv Real-Time SDK can connect to this distribution system. |
| Reactor | The Reactor is a low-level, open-source, easy-to-use layer above the Enterprise Transport API. It offers heartbeat management, connection and item recovery, and many other features to help simplify application code for users. |
| RMTES | A multi-lingual text encoding standard |
| RSSL | Refinitiv Source Sink Library |
| RTT | Round Trip Time, this definition is used for round trip latency monitoring feature. |
| RWF | Refinitiv Wire Format, a Refinitiv proprietary binary format for data representation. |
| RDF-D | Refinitiv Data Feed Direct |
| UML | Unified Modeling Language |
| UTF-8 | 8-bit Unicode Transformation Format |

Table 1: Acronyms and Abbreviations

1.4 References

1. Enterprise Message API Java Edition *Refinitiv Domain Model Usage Guide*
2. *API Concepts Guide*
3. *Enterprise Message API Java Edition Reference Manual*
4. Enterprise Message API Java Edition *Developers Guide*
5. The [Refinitiv Developer Community](#)

1.5 Documentation Feedback

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

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

1.6 Document Conventions

This document uses the following types of conventions:

- Typographic
- Field and Text Values
- Boolean Values

1.6.1 Typographic

This document uses the following types of conventions:

- Java classes, methods, in-line code snippets, and types are shown in **Courier New** font.
- Parameters, filenames, tools, utilities, and directories are shown in **Bold** font.
- Document titles and variable values are shown in *italics*.
- When initially introduced, concepts are shown in ***Bold, Italics***.
- Longer code examples are shown in Courier New font against a gray background. For example:

```
AppClient client = new AppClient();

OmmConsumerConfig config = EmaFactory.createOmmConsumerConfig();

OmmConsumer consumer =
    EmaFactory.createOmmConsumer(config.operationModel(OperationModel.USER_DISPATCH)
        .host("localhost:14002").username("user"));

ReqMsg reqMsg = EmaFactory.createReqMsg();

consumer.registerClient(reqMsg.domainType(EmaRdm.MMT_MARKET_BY_PRICE).serviceName(
    "DIRECT_FEED").name("BBH.ITS"), client);
```

1.6.2 Field and Text Values

The value for individual fields in XML files are specified as **<fieldName value="field_value"/>** where:

- **fieldName** is the name of the field and cannot contain white space.
- **field_value** sets the field's value and is always included in double quotes.

NOTE: Except for examples, double quotes are omitted from the field (parameter) descriptions throughout the remainder of this document.

Though enumerations have text values (i.e., SOCKET), in the software, text values are represented as numbers (required for programmatic configuration). When introduced, enumerations are listed along with their textual values.

1.6.3 Boolean Values

When configuring a Boolean expression, you can use any number; however EMA interprets such expressions in the following manner:

- **0** (or any other value): false
- **1**: true

2 EMA Configuration General Overview

2.1 About Message API Configuration

You write the Message API configuration using a simple XML schema, some settings of which can be changed via software function calls. The initial configuration compiled into the Message API software defines a minimal set of configuration parameters. Message API users can also supply their own custom XML file (e.g., **EmaConfig.xml**) to specify configuration parameters. For details on deploying a custom XML file, refer to Section 4.3.1. Additionally, programmatic interfaces can change parameter settings.

Message API configuration data is divided into the following groups:

- **Consumer:** Consumer configuration data are the highest-level description of the application. Such settings typically select entries from the channel, and dictionary groups.
- **Provider:** Where *Provider* is either an IProvider or NiProvider. *Provider* configuration data is the highest-level description of the application. Such settings typically select entries from the channel (NiProvider only), and directory groups.
- **Channel:** Channel configuration data describe various connection alternatives and provide configuration alternatives for those connections.
- **Dictionary:** Dictionary configuration data set the location information for dictionary alternatives.
- **Directory:** Directory configuration data configure source directory refresh information.

The Consumer and *Provider* groups are top-level configuration groups. Specific consumer and provider applications select their configurations according to the name specified in the **consumerName ()** or **providerName ()** method (for details on these methods, refer to Section 4.4.1).

This manual discusses the six configuration groups and the configuration parameters available to each group.

2.2 Parameter Overview

Many default behaviors are hard-coded into the Enterprise Message API library and globally enforced. However, if you need to change API behaviors or configure the API for your specific deployment, you can use the Enterprise Message API's XML configuration file (**EmaConfig.xml**) and adjust behaviors using the appropriate parameters (discussed in this section). While the Enterprise Message API globally enforces a set of default behaviors, certain other default behaviors are dependent on the use of the XML file and its settings.

For a list of default behaviors (and the parameters that you can use to change these behaviors) refer to Section 2.3.

2.3 Default Behaviors

When the Enterprise Message API library needs a parameter, it behaves according to its hard coded configuration. You can change the APIs behavior by providing a valid alternate value either through the use of **EmaConfig.xml**, function calls, or programmatic methods.

| PARAMETER | TYPE | DEFAULT BEHAVIOR | NOTES |
|----------------------------|--------|--|---|
| Host | String | localhost | Specifies the host name of the server to which the application connects. The parameter value can be a remote host name or IP address. |
| Port | String | 14002 (for consumers) 14003 (for non-interactive providers) | Specifies the port number on the server to which the application connects. |
| DefaultConsumer | String | EmaConsumer | If consumer components are configured, the Enterprise Message API ignores this parameter. |
| DefaultIProvider | String | EmailProvider | If interactive provider components are configured, the Enterprise Message API ignores this parameter. |
| DefaultNiProvider | String | EmaNiProvider | If non-interactive provider components are configured, the Enterprise Message API ignores this parameter. |
| RdmFieldDictionaryFileName | String | ./RDMFieldDictionary | Specifies the path and name of the RdmFieldDictionary file. |
| EnumTypeDefFileName | String | ./enumtype.def | Specifies the path and name of the enumtypeDef dictionary file. |

Table 2: Global Configuration

3 Configuration Groups

3.1 ConsumerGroup

A **ConsumerGroup** contains two elements:

- A **DefaultConsumer** element, which you can use to specify a default **Consumer** component. If a default **Consumer** is not specified in the **ConsumerGroup**, the Enterprise Message API uses the first Consumer listed in the **ConsumerList**. For details on configuring a default **Consumer**, refer to Section 3.1.2.
- A **ConsumerList** element, which contains one or more **Consumer** components (each should be uniquely identified by a `<Name .../>` entry). The consumer component is the highest-level abstraction within an application and typically refers to **Channel** and/or **Dictionary** components which specify consumer capabilities.

For a generic **ConsumerGroup** XML schema, refer to Section 3.1.1.

For details on configuring a **ConsumerGroup**, refer to Section 3.1.3.

For a list of parameters you can use in configuring a **Consumer**, refer to Section 3.1.4.

3.1.1 Generic XML Schema for ConsumerGroup

The generic XML schema for **ConsumerGroup** is as follows:

```
<ConsumerGroup>
  <DefaultConsumer value="VALUE"/>
  <ConsumerList>
    <Consumer>
      <Name value="VALUE"/>
      ...
    </Consumer>
  </ConsumerList>
</ConsumerGroup>
```

3.1.2 Setting a Default Consumer

If a **DefaultConsumer** is not specified, then the Enterprise Message API uses the first **Consumer** component in the **ConsumerGroup**. However, you can specify a default consumer by including the following parameter on a unique line inside **ConsumerGroup** but outside **ConsumerList**.

```
<DefaultConsumer value="VALUE"/>
```

3.1.3 Configuring Consumers in a ConsumerGroup

To configure a **Consumer** component, add the appropriate parameters to the target consumer in the XML schema, each on a unique line (for a list of available **Consumer** parameters, refer to Section 3.1.4).

For example, if your configuration includes channel schemas, you specify the desired channel schema by adding the following parameter inside the appropriate **Consumer** section:

```
<Channel value="VALUE"/>
```

Consumer components can use different channel schemas if the configuration includes more than one.

3.1.4 Consumer Entry Parameters

Use the following parameters when configuring a **Consumer**.

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------------|--------|---------|---|
| CatchUnknownJsonFids | int | 1 | Specifies whether the RWF/JSON conversion catches unknown JSON field IDs. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not catch unknown JSON field IDs. 1 (true): Catch unknown JSON field IDs. |
| CatchUnknownJsonKeys | int | 0 | Specifies whether the RWF/JSON conversion catches unknown JSON keys. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not catch unknown JSON keys. 1 (true): Catch unknown JSON keys. |
| Channel | String | N/A | Specifies the channel that the Consumer component should use. This channel must match the Name parameter from the appropriate <Channel> entry in the ChannelGroup configuration. If Channel is not specified, the Enterprise Message API resorts to default channel behavior when needed. For further details on the <Channel> entry and default behaviors, refer to Section 3.3. |
| ChannelSet | String | N/A | Specifies a comma-separated set of channels names. Each listed channel name should have an appropriate <Channel> entry in the ChannelGroup . Channels in the set will be tried with each reconnection attempt until a successful connection is made. For further details refer to Section 3.3.7. NOTE: If both Channel and ChannelSet are configured, then the Enterprise Message API uses the parameter that is configured last in the file. For example, if <Channel> is configured after <ChannelSet> then the Enterprise Message API uses <Channel> , but if <ChannelSet> is configured after <Channel> then the Enterprise Message API uses <ChannelSet> . |
| CloseChannelFromConverterFailure | int | 1 | Specifies that the Enterprise Message API should close the channel if the Enterprise Message API fails to parse JSON messages or if the Enterprise Message API receives JSON error messages. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not close the channel. 1 (true): Close the channel. |

Table 3: Consumer Group Parameters

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|--------------------------|--------|---------|---|
| DefaultServiceID | int | 1 | Specifies a default service ID for RWF/JSON conversion if both service name and ID are missing. The maximum allowable value is 65535 . |
| Dictionary | String | N/A | Specifies how the consumer should access its dictionaries (it must match the Name parameter from the appropriate <Dictionary> entry in the DictionaryGroup configuration). If Dictionary is not specified, the Enterprise Message API uses the channel's dictionary when needed. For further details on this default behavior, refer to Section 3.5. |
| DictionaryRequestTimeout | long | 45,000 | Specifies the amount of time (in milliseconds) the application has to download dictionaries from a provider before the OmmConsumer throws an exception. If set to 0 , the Enterprise Message API will wait for a response indefinitely. NOTE: If ChannelSet is configured: <ul style="list-style-type: none"> The Enterprise Message API honors DictionaryRequestTimeout only on its first connection. If the channel supporting the first connection goes down, the Enterprise Message API does not use DictionaryRequestTimeout on subsequent connections. |
| DirectoryRequestTimeout | long | 45,000 | Specifies the amount of time (in milliseconds) the provider has to respond with a source directory refresh message before the OmmConsumer throws an exception. If set to 0 , the Enterprise Message API will wait for a response indefinitely. NOTE: If ChannelSet is configured: <ul style="list-style-type: none"> The Enterprise Message API honors DirectoryRequestTimeout only on its first connection. If the channel supporting the first connection goes down, the Enterprise Message API does not use DirectoryRequestTimeout on subsequent connections. |
| DispatchTimeoutApiThread | int | 0 | Specifies the duration (in microseconds) for which the internal Enterprise Message API thread is inactive before going active to check whether a message was received. If set to zero, the Enterprise Message API internal thread goes active only if it gets notified about a received message. |
| EnableRtt | long | 0 | Specifies whether the OmmConsumer supports gathering RoundTripLatency statistics. If enabled, the Watchlist handles automatic processing of RTT requests sent by the provider. EnableRtt expresses the consumer's consent to process RTT requests. The provider may choose either to send or not to send the requests at its own discretion. Available values include: <ul style="list-style-type: none"> 0 (false) Any value > 0 (true) |
| ItemCountHint | long | 100,000 | Specifies the number of items the application expects to request. If set to 0 , the Enterprise Message API resets it to 1024 . For better performance, the application can set this to the approximate number of item requests it expects. |

Table 3: Consumer Group Parameters (Continued)

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------|--------|---------|--|
| JsonExpandedEnumFields | int | 0 | Sets the RWF/JSON conversion to expand enumerated values in field entries to their display values for JSON protocol. Possible values are: <ul style="list-style-type: none"> • 0 (false): Do not expand enumerated fields. • 1 (true): Expand enumerated fields. |
| LoginRequestTimeout | long | 45,000 | Specifies the amount of time (in milliseconds) the provider has to respond with a login refresh message before the OmmConsumer throws an exception. If set to 0 , the Enterprise Message API will wait for a response indefinitely. NOTE: If ChannelSet is configured: <ul style="list-style-type: none"> • The Enterprise Message API honors LoginRequestTimeout only on its first connection. • If the channel supporting the first connection goes down, the Enterprise Message API does not use LoginRequestTimeout on subsequent connections. |
| MaxDispatchCountApiThread | long | 100 | Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break. |
| MaxDispatchCountUserThread | long | 100 | Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the OmmConsumer::dispatch() . |
| MaxOutstandingPosts | long | 100,000 | Specifies the maximum allowable number of on-stream posts waiting for an acknowledgment before the OmmConsumer disconnects. |
| MaxEventsInPool | int | -1 | Specifies the maximum number of event objects in the event's pool. |
| MsgKeyInUpdates | int | 1 | Specifies whether the Enterprise Message API fills in message key values on updates using the message key provided with the request. Available values include: <ul style="list-style-type: none"> • 0 (false): Do not fill in the message's key values (values received from the wire are preserved). • 1 (true): Fill in the message's key values (values received from the wire are overridden). |
| Name | String | N/A | Specifies the name of this Consumer component. Name is required when creating a Consumer component. You can use any value for Name . |
| ObeyOpenWindow | int | 1 | Specifies whether the OmmConsumer obeys the OpenWindow from services advertised in a provider's Source Directory response. Available values include: <ul style="list-style-type: none"> • 0 (false) • 1 (true) |
| PostAckTimeout | long | 15,000 | Specifies the length of time (in milliseconds) a stream waits to receive an ACK for an outstanding post before forwarding a negative acknowledgment to the application. If set to 0 , the Enterprise Message API will wait for a response indefinitely. |
| ReconnectAttemptLimit | int | -1 | Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel when it fails. If set to -1 , the consumer and non-interactive provider continually attempt to reconnect. |

Table 3: Consumer Group Parameters (Continued)

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-----------------------------|--------|---------|--|
| ReconnectMaxDelay | int | 5000 | Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the ReconnectMinDelay parameter. |
| ReconnectMinDelay | int | 1000 | Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from reconnectMinDelay to reconnectMaxDelay . |
| ReissueTokenAttemptInterval | int | 5000 | Sets the delay (in milliseconds) before the OMMConsumer attempts to reissue the token. The minimum interval is 1000 milliseconds, while the default setting is 5000. |
| ReissueTokenAttemptLimit | int | -1 | Specifies the maximum number of times the OMMConsumer attempts to reissue the token. If set to default (i.e., -1), there is no maximum limit. |
| RequestTimeout | long | 15,000 | Specifies the amount of time (in milliseconds) the OmmConsumer waits for a response to a request before sending another request. If set to 0 , the Enterprise Message API will wait for a response indefinitely. |
| RestRequestTimeOut | long | 45000 | Specifies the timeout (in milliseconds) for token service and service discovery request. If the request times out, the OMMConsumer resends the token reissue and the timeout restarts. If the request times out, the OMMConsumer does not retry. If set to 0 , there is no timeout. |
| ServiceCountHint | long | 513 | Sets the size of directory structures for managing services. If the application specifies 0 , the Enterprise Message API resets it to 513 . |
| TokenReissueRatio | Double | .8 | Specifies the ratio with which to multiply the access token's expiration time (in seconds) to determine the length of time the OMMConsumer waits before retrieving a new access token and refreshing its connection to Refinitiv Real-Time - Optimized. The valid range is from 0.05 to 0.95 . |
| XmlTraceToStdout | int | 0 | Specifies whether the Enterprise Message API traces its messages in XML format to stdout. Possible values are: <ul style="list-style-type: none"> 0 (false): Turns off tracing. 1 (true): Turns on tracing to stdout. |

Table 3: Consumer Group Parameters (Continued)

3.2 Provider Groups

The Enterprise Message API supports both interactive and non-interactive provider groups. The type of provider you configure will determine the group names and parameters that you use. To simplify the content in this guide, parameters and names use the variable ***Provider***. Throughout this section (and the remainder of the manual), the value for ***Provider*** is dependent on the type of provider which you configure:

- For non-interactive providers, ***Provider*** is **NiProvider**.
- For interactive providers, ***Provider*** is **IProvider**.

A ***ProviderGroup*** contains two elements:

- A **DefaultProvider** element, which you can use to specify a default **NiProvider** component. If a default ***Provider*** is not specified in the ***ProviderGroup***, The Enterprise Message API uses the first non-interactive provider listed in the ***ProviderList***. For details on configuring a default ***Provider***, refer to Section 3.2.2.
- A ***ProviderList*** element, which contains one or more ***Provider*** components (each should be uniquely identified by a **<Name .../>** entry). The non-interactive provider component is the highest-level abstraction within an application and typically refers to **Channel** (used by non-interactive providers) **Server** (used by interactive providers) and/or **Directory** components which specify provider capabilities.

For a generic ***ProviderGroup*** XML schema, refer to Section 3.2.1.

For details on configuring a ***ProviderGroup***, refer to Section 3.2.3.

For a list of parameters you can use in configuring a ***Provider***, refer to Section 3.2.4.

3.2.1 Generic XML Schema for Provider Group

The generic XML schema for ***ProviderGroup*** is as follows:

```
<ProviderGroup>
  <DefaultProvider value="VALUE"/>
  <ProviderList>
    <Provider>
      <Name value="VALUE"/>
      ...
    </Provider>
  </ProviderList>
</ProviderGroup>
```

3.2.2 Setting a Default Provider

If a **DefaultProvider** is not specified, then the Enterprise Message API uses the first ***Provider*** component in the ***ProviderGroup***. However, you can specify a default provider by including the following parameter on a unique line inside ***ProviderGroup*** but outside ***ProviderList***.

```
<DefaultProvider value="VALUE"/>
```

3.2.3 Configuring a *Provider* in a *ProviderGroup*

To configure a ***Provider*** component, add the appropriate parameters to the `target` in the XML schema, each on a unique line (for a list of available ***Provider*** parameters, refer to Section 3.2.4).

For example, if your configuration includes channel schemas, you specify the desired channel schema by adding the following parameter inside the appropriate ***Provider*** section:

```
<Channel value="VALUE"/>
```

If your provider component needs more than one channel schema, you can configure each unique schema in the XML file.

3.2.4 Provider Entry Parameters

Use the following parameters when configuring a ***Provider***. Certain parameters can only be used with a specific provider type (e.g., ***Channel*** can only be used with an ***NiProvider***). The parameter's description will mention any provider-type restrictions.

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|---------------------------------------|------|---------|---|
| AcceptDirMessageWithoutMinFilters | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming directory request messages without the minimum required INFO and STATE directory filters. |
| AcceptMessageSameKeyButDiffStream | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming request messages even though they have a message key, domain, and private stream flag that match those of an existing request which uses a different stream ID. |
| AcceptMessageThatChangesService | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming request messages for reissuing the service name of an existing item stream. |
| AcceptMessageWithoutAcceptingRequests | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming request messages even though the source directory is not accepting requests. |
| AcceptMessageWithoutBeingLogin | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming request messages even though the interactive provider has not accepted a login request. |
| AcceptMessageWithoutQosInRange | int | 0 | Used only with <i>IPProvider</i> . Sets the <i>IPProvider</i> to accept incoming request messages even though the requesting QoS is not in the QoS range of the source directory. |
| CatchUnknownJsonFids | int | 1 | Specifies whether the RWF/JSON conversion catches unknown JSON field IDs. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not catch unknown JSON field IDs. 1 (true): Catch unknown JSON field IDs. |

Table 4: *ProviderGroup* Parameters

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------------|--------|---------|--|
| CatchUnknownJsonKeys | int | 0 | Specifies whether the RWF/JSON conversion catches unknown JSON keys. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not catch unknown JSON keys. 1 (true): Catch unknown JSON keys. |
| Channel | String | N/A | Used only with NiProvider . Specifies the channel that the NiProvider component should use. This channel must match the Name parameter from the appropriate <Channel> entry in the ChannelGroup configuration. If Channel is not specified, the Enterprise Message API resorts to default channel behavior when needed. For further details on the <Channel> entry and default behaviors, refer to Section 3.3. |
| ChannelSet | String | N/A | Used only with NiProvider . Specifies a comma-separated set of channel names. Each channel name must have a corresponding <Channel> entry in the ChannelGroup . In the event of a reconnection, Channels in the set are tried until a successful connection is made. For further details, refer to Section 3.3.7. NOTE: If both Channel and ChannelSet are configured, the Enterprise Message API uses the parameter configured last (linearly) in the file. For example: <ul style="list-style-type: none"> If <Channel> is configured after <ChannelSet>, the Enterprise Message API uses <Channel>. If <ChannelSet> is configured after <Channel>, the Enterprise Message API uses <ChannelSet>. |
| CloseChannelFromConverterFailure | int | 1 | Specifies that the Enterprise Message API should close the channel if it fails to parse JSON messages or if it receives JSON error messages. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not close the channel. 1 (true): Close the channel. |
| DefaultServiceID | int | 1 | Specifies a default service ID for RWF/JSON conversion if both service name and ID are missing. The maximum allowable value is 65535 . |
| DispatchTimeoutApiThread | int | 0 | Specifies the duration (in microseconds) for which the internal Enterprise Message API thread is inactive before going active to check whether a message was received. If set to zero, the thread goes active only if notified about a received message. |
| EnforceAckIDValidation | long | 0 | Used only with IProvider . Specifies whether IProvider has to validate the AckId attribute when an AckMsg calls OmmProvider::submit() . If validation is turned on, then AckId must be equal to the PostId of PostMsg received by the IProvider . Available values include: <ul style="list-style-type: none"> 1 (true): Validate the AckId. 0 (false): Do not validate the AckId. |

Table 4: *ProviderGroup* Parameters (Continued)

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-----------------------------|--------|---------|---|
| EnumTypeFragmentSize | int | 128000 | Used only with IProvider . Sets the maximum enumeration types fragmentation size (in bytes) for each multi-part refresh message. |
| FieldDictionaryFragmentSize | int | 8192 | Used only with IProvider . Sets the maximum field dictionary fragmentation size (in bytes) for each multi-part refresh message. |
| ItemCountHint | long | 100,000 | Specifies the number of items the application expects to maintain. If set to 0 , the Enterprise Message API resets it to 1024 . For better performance, the application can set this to the approximate number of items it maintains. |
| JsonExpandedEnumFields | int | 0 | Sets the RWF/JSON conversion to expand enumerated values in field entries to their display values for JSON protocol. Possible values are: <ul style="list-style-type: none"> 0 (false): Do not expand enumerated fields. 1 (true): Expand enumerated fields. |
| LoginRequestTimeout | long | 45,000 | Used only with NiProvider . Specifies the amount of time (in milliseconds) the provider has to respond with a login refresh message before the OmmProvider throws an exception. If set to 0 , the Enterprise Message API will wait for a response indefinitely. NOTE: When ChannelSet is configured, the Enterprise Message API honors LoginRequestTimeout only on its first connection. If the channel supporting the first connection goes down, the Enterprise Message API does not use LoginRequestTimeout on subsequent connections. |
| MaxDispatchCountApiThread | long | 100 | Specifies the maximum number of messages the Enterprise Message API dispatches before taking a real-time break. |
| MaxDispatchCountUserThread | long | 100 | Specifies the maximum number of messages the Enterprise Message API can dispatch in a single call to the OmmProvider::dispatch() . |
| MergeSourceDirectoryStreams | long | 1 | Used only with NiProvider . Specifies whether the Enterprise Message API merges all source directory streams (configured and user-submitted) into one stream: <ul style="list-style-type: none"> 1 (true) 0 (false) |
| Name | String | N/A | Specifies the name of this Provider component. Name is required when creating a Provider component. You can use any value for Name . |
| ReconnectAttemptLimit | int | -1 | Used only with NiProvider . Specifies the maximum number of times the consumer and non-interactive provider attempt to reconnect to a channel to a channel when it fails. If set to -1 , the consumer and non-interactive provider continually attempt to reconnect. |

Table 4: **ProviderGroup** Parameters (Continued)

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------------|--------|---------|--|
| ReconnectMaxDelay | int | 5000 | Used only with NiProvider . Sets the maximum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. Refer also to the ReconnectMinDelay parameter. |
| ReconnectMinDelay | int | 1000 | Used only with NiProvider . Specifies the minimum amount of time the consumer and non-interactive provider wait (in milliseconds) before attempting to reconnect a failed channel. This wait time increases with each connection attempt, from reconnectMinDelay to reconnectMaxDelay . |
| RecoverUserSubmitSourceDirectory | long | 1 | Used only with NiProvider . Specifies whether the Enterprise Message API recovers user-submitted source directories when recovering from a disconnect: <ul style="list-style-type: none"> • 1 (true) • 0 (false) |
| RefreshFirstRequired | long | 1 | Specifies whether the Enterprise Message API requires the application to send a refresh message prior to sending update messages. Available values include: <ul style="list-style-type: none"> • 1 (true) • 0 (false) |
| RemoveItemsOnDisconnect | long | 1 | Used only with NiProvider . Specifies whether the Enterprise Message API removes items from its internal hash table whenever it disconnects from the Refinitiv Real-Time Advanced Data Hub: <ul style="list-style-type: none"> • 1 (true) • 0 (false) |
| RequestTimeout | long | 15000 | Specifies the length of time (in milliseconds) the OmmProvider waits for a response to a request before sending another request (the DICTIONARY domain will not send another request). If set to 0 , the Message API waits for a response indefinitely. |
| Server | String | N/A | Used only with IPProvider . Specifies the channel that the IPProvider component should use. This channel must match the Name parameter from the appropriate <Server> entry in the ServerGroup configuration. If Server is not specified, the Enterprise Message API resorts to default channel behavior when needed. For further details on the <Server> entry and default behaviors, refer to Section 3.4. |

Table 4: *ProviderGroup* Parameters (Continued)

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|------------------|------|---------|--|
| ServiceCountHint | long | 513 | Sets the size of directory structures for managing services. If the application specifies 0 , the Enterprise Message API resets it to 513 . |
| XmlTraceToStdout | int | 0 | Specifies whether the Enterprise Message API traces its messages in XML format to stdout. Possible values are: <ul style="list-style-type: none">• 0 (false): Turns off tracing.• 1 (true): Turns on tracing to stdout. |

Table 4: *ProviderGroup* Parameters (Continued)

3.3 Channel Group

ChannelGroup is used only with the **Consumer** and **NiProvider**.

The **ChannelGroup** contains a **ChannelList**, which contains one or more **Channel** entries (each uniquely identified by a **<Name .../>** entry). Each channel includes a set of connection parameters for a specific connection or connection type.

There is no default channel. If an Enterprise Message API application needs a specific channel, you must specify this in the appropriate **Consumer** or **NiProvider** section.

- For details on the parameters you can use to configure the **Consumer** component, refer to Section 3.1.4.
- For details on the parameters you can use to configure the **NiProvider** component, refer to Section 3.2.4
- For a generic **ChannelGroup** XML schema, refer to Section 3.3.1.
- For a list of universal parameters you can use in configuring any type of **Channel** regardless of the channel type, refer to Section 3.3.2.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_SOCKET**, refer to Section 3.3.3.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_ENCRYPTED**, refer to Section 3.3.4.
- For a list of parameters you can use only when configuring a **Channel** whose channel type is **RSSL_HTTP**, refer to Section 3.3.4.

3.3.1 Generic XML Schema for ChannelGroup

The top-level XML schema for the **ChannelGroup** is as follows:

```
<ChannelGroup>
  <ChannelList>
    <Channel>
      <Name value="VALUE"/>
      ...
    </Channel>
  </ChannelList>
</ChannelGroup>
```

3.3.2 Universal Channel Entry Parameters

You can use the following parameters in any **<Channel>** entry, regardless of the **ChannelType**.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|-------------------------|--------|--|--|
| ChannelType | String | RSSL_SOCKET | <p>Specifies the type of channel or connection used to connect to the server.</p> <p>Calling the host function can change this field. For details on this event, refer to Section 4.4.2.</p> <p>Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5).</p> <p>Available values include:</p> <ul style="list-style-type: none"> • RSSL_SOCKET • RSSL_ENCRYPTED • RSSL_HTTP • WEBSOCKET |
| ConnectionPingTimeout | long | 30000 | Specifies the duration (in milliseconds) after which the Enterprise Message API terminates the connection if it does not receive communication or pings from the server. |
| EnableSessionManagement | long | 0 | <p>Specifies whether the channel manages the authentication token on behalf of the user used to keep the session alive. If set to 1, the channel obtains the authentication token and refreshes it on behalf of user to keep session active. The default setting is 0. You can use this parameter only in with Enterprise Message API consumers.</p> <p>You can use EnableSessionManagement only on an RSSL_ENCRYPTED ChannelType.</p> |
| GuaranteedOutputBuffers | long | 100 | <p>Specifies the number of guaranteed buffers (allocated at initialization time) available for use by each RsslChannel when writing data. Each buffer is created to contain maxFragmentSize bytes.</p> <p>For details on RsslChannel and maxFragmentSize, refer to the <i>Transport API Developers Guide</i>.</p> |
| HighWaterMark | long | 6144 | Specifies the upper buffer-usage threshold for the channel. |
| InitializationTimeout | long | 5 (10 when used with RSSL_ENCRYPTED ChannelType) | Specifies the time (in seconds) to wait for the successful initialization of a channel. |
| InterfaceName | String | "" | <p>Specifies a character representation of the IP address or hostname of the local network interface over which the Enterprise Message API sends and receives content.</p> <p>InterfaceName is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used.</p> |

Table 5: Universal <Channel> Parameters

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|-----------------|--------|-----------|---|
| Location | String | us-east-1 | Used only when host and port are unspecified, Location specifies the cloud location of the service provider endpoint to which the RTSDK API establishes a connection. If Location is not specified, the default setting is us-east-1 . In any particular cloud location, the Enterprise Message API connects to the endpoint that provides two available zones for the location (e.g., [us-east-1a , us-east-1b]). You can use Location only on an RSSL_ENCRYPTED ChannelType . |
| Name | String | | Specifies the Channel 's name. |
| NumInputBuffers | long | 10 | Specifies the number of buffers used to read data. Buffers are sized according to maxFragmentSize . For details on RsslChannel and maxFragmentSize , refer to the <i>Transport API Developers Guide</i> . |
| SysRecvBufSize | long | 0 | Specifies the size (in bytes) of the system's receive buffer for this channel. For exact, effective values, refer to your operating system documentation.. |
| SysSendBufSize | long | 0 | Specifies the size (in bytes) of the system's send buffer for this channel. For exact, effective values, refer to your operating system documentation.. |

Table 5: Universal <Channel> Parameters (Continued)

3.3.3 Parameters for Use with Channel Type: RSSL_SOCKET

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is **RSSL_SOCKET**.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|----------------------|--------|-----------|---|
| CompressionThreshold | long | 30 | Sets the message size threshold (in bytes, the allowed value is 30-Integer.MAX_VALUE), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size. |
| CompressionType | String | None | Specifies the Enterprise Message API's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include: <ul style="list-style-type: none"> • None • ZLib • LZ4 |
| | | | NOTE: A server can be configured to force a particular compression type, regardless of client settings. |
| DirectWrite | int | 0 | Specifies whether to set the direct socket write flag when sending data on a channel. When the flag is set, every package is sent on the wire immediately on the submit call. If direct write is not set, the package might be placed into an internal queue which is later flushed onto the wire. Possible values are: <ul style="list-style-type: none"> • 0: Send data without the direct socket write flag. • 1: Send data with the direct socket write flag. |
| Host | String | localhost | Specifies the host name of the server to which the Enterprise Message API connects. The parameter value can be a remote host name or IP address. |
| Port | String | 14002 | Specifies the port on the remote server to which the Enterprise Message API connects. |
| ProxyHost | String | "" | Specifies the host name of the proxy to which the Enterprise Message API connects. The parameter value can be a host name or an IP address. Any value provided by a function call overrides the setting in configuration file. |
| ProxyPort | String | "" | Specifies the port on the proxy to which the Enterprise Message API connects. Any value provided by a function call overrides the setting in configuration file. |
| TcpNodelay | int | 1 | Specifies whether to use Nagle's algorithm when sending data. Available values are: <ul style="list-style-type: none"> • 0: Send data using Nagle's algorithm. • 1: Send data without delay. |

Table 6: Parameters for Channel Type: RSSL_SOCKET

3.3.4 Parameters for Use with Channel Types: RSSL_HTTP

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is **RSSL_HTTP**.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|----------------------|--------|-----------|--|
| CompressionThreshold | int | 30 | Sets the message size threshold (in bytes, the allowed value is 30-UInt32 MAX), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size. |
| CompressionType | String | None | Specifies the Enterprise Message API's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level. Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include: <ul style="list-style-type: none"> • None • ZLib • LZ4 |
| | | | NOTE: A server can be configured to force a particular compression type, regardless of client settings. |
| Host | String | localhost | Specifies the host name of the server to which the Enterprise Message API connects. The parameter value can be a remote host name or IP address. |
| ObjectName | String | "" | Specifies the object name to pass along with the underlying URL in HTTP and HTTPS connection messages. |
| Port | String | 14002 | Specifies the port on the remote server to which the Enterprise Message API connects. |
| ProxyHost | String | "" | Specifies the host name of the proxy to which the Enterprise Message API connects. The parameter value can be a host name or an IP address. Any value provided by a function call overrides the setting in configuration file. |
| ProxyPort | String | "" | Specifies the port on the proxy to which the Enterprise Message API connects. Any value provided by a function call overrides the setting in configuration file. |
| TcpNodelay | int | 1 | Specifies whether to use Nagle's algorithm when sending data. Available values are: <ul style="list-style-type: none"> • 0: Send data using Nagle's algorithm. • 1: Send data without delay. |

Table 7: Parameters for Channel Type: RSSL_HTTP

3.3.5 Parameters for Use with Channel Types: WEBSOCKET

In addition to the universal parameters listed in Section 3.3.2, you can use the following parameters to configure a channel whose type is **WEBSOCKET**. A **WEBSOCKET** channel does not support the LZ4 compression type.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|----------------|--------|----------------------------------|--|
| WsMaxMsgSize | int | 61440 | Specifies the maximum size of messages that the WebSocket transport can send or read. |
| WsProtocols | String | tr_json2, rssl.rwf, rssl.json.v2 | Specifies a list of supported/preferred protocols in order of preference from highest to lowest. |

Table 8: Parameters for Channel Types: WEBSOCKET

3.3.6 Parameters for Use with Channel Types: RSSL_ENCRYPTED

In addition to the universal parameters listed in Section 3.3.2, and the parameters listed in the section specific to the protocol type you use (i.e., Section 3.3.3 for socket connections or Section 3.3.4 for HTTP connections), use the following parameters to configure a channel whose type is **RSSL_ENCRYPTED**.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|-----------------------|--------|-----------|---|
| EncryptedProtocolType | String | RSSL_HTTP | Specifies the type of protocol used for this encrypted connection. <ul style="list-style-type: none"> • RSSL_SOCKET (0) • RSSL_HTTP (2) • WEBSOCKET (7) |

Table 9: Parameters for Channel Types: RSSL_ENCRYPTED

3.3.7 Example XML Schema for Configuring ChannelSet

The following is an example XML schema for use in configuring a **ChannelSet**:

```
<ConsumerGroup>
  <ConsumerList>
    <Consumer>
      ...
      <!-- ChannelSet specifies an ordered list of Channels to which OmmConsumer will attempt -->
      <!-- to connect, one at a time, if the previous one fails to connect -->
      <ChannelSet value="VALUE1, VALUE2, ..."/>
      ...
    </Consumer>
  </ConsumerList>
</ConsumerGroup>
```

3.3.8 Example ChannelSet XML Configuration

The following XML example illustrates a specific ChannelSet configuration using the XML schema introduced in Section 3.3.7:

```
<ConsumerGroup>
  <ConsumerList>
    <Consumer>
      <Name value="Consumer_1"/>
      <!-- ChannelSet specifies an ordered list of Channels to which OmmConsumer will attempt -->
      <!-- to connect, one at a time, if the previous one fails to connect -->
      <ChannelSet value="Channel_1, Channel_2"/>
      <ReconnectAttemptLimit value="10"/>
      <XmlTraceToStdout value="1"/>
    </Consumer>
  </ConsumerList>
</ConsumerGroup>
<ChannelGroup>
  <ChannelList>
    <Channel>
      <Name value="Channel_1"/>
      <ChannelType value="ChannelType::RSSL_SOCKET"/>
      <Host value="localhost"/>
      <Port value="14002"/>
    </Channel>
    <Channel>
      <Name value="Channel_2"/>
      <ChannelType value="ChannelType::RSSL_SOCKET"/>
      <Host value=" localhost "/>
      <Port value="14008"/>
    </Channel>
  </ChannelList>
</ChannelGroup>
```

3.3.9 Example Programmatic Configuration for ChannelSet

The following XML example illustrates a programmatic ChannelSet configuration in Java:

```
Map innerMap = EmaFactory.createMap();
Map configMap = EmaFactory.createMap();
ElementList elementList = EmaFactory.createElementList();
ElementList innerElementList = EmaFactory.createElementList();

elementList.add(EmaFactory.createElementEntry().ascii("DefaultConsumer", "Consumer_1"));
innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelSet", "Channel_1", Channel_2));
innerElementList.add(EmaFactory.createElementEntry().ascii("Dictionary", "Dictionary_1"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Consumer_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map( "ConsumerList", innerMap ));
innerMap.clear();
configMap.add(EmaFactory.createMapEntry().keyAscii( "ConsumerGroup", MapEntry.MapAction.ADD,
    elementList ));
elementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelType",
    "ChannelType::RSSL_SOCKET"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Host", "localhost"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Port", "14002"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Channel_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelType",
    "ChannelType::RSSL_SOCKET"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Host", "121.1.1.100"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Port", "14008"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Channel_2", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map( "ChannelList", innerMap ));
innerMap.clear();

configMap.add(EmaFactory.createMapEntry().keyAscii( "ChannelGroup", MapEntry.MapAction.ADD,
    elementList ));
elementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("DictionaryType",
    "DictionaryType::ChannelDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("RdmFieldDictionaryFileName",
    "./RDMFieldDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("EnumTypeDefFileName", "./enumtype.def));
```

```
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Dictionary_1", MapEntry.MapAction.ADD,  
    innerElementList));  
innerElementList.clear();  
  
elementList.add(EmaFactory.createElementEntry().map( "DictionaryList", innerMap ));  
configMap.add(EmaFactory.createMapEntry().keyAscii( "DictionaryGroup", MapEntry.MapAction.ADD,  
    elementList ));  
elementList.clear();
```

3.4 Server Group

ServerGroup is used only with an **IProvider**.

The **ServerGroup** contains a **ServerList**, which contains one or more **Server** entries (each uniquely identified by a **<Name .../>** entry). Each channel includes a set of connection parameters for a specific connection or connection type.

There is no default server. If an Enterprise Message API application needs a specific server, you need to specify this in the appropriate **Consumer** or **IProvider** section.

- For details on the parameters you can use to configure the **Consumer** component, refer to Section 3.1.4.
- For details on the parameters you can use to configure the **IProvider** component, refer to Section 3.2.4.
- For a generic **ServerGroup** XML schema, refer to Section 3.4.1.
- For a list of parameters you can use in configuring **Server**, refer to Section 3.4.2.

3.4.1 Generic XML Schema for ServerGroup

The top-level XML schema for the **ServerGroup** is as follows:

```
<ServerGroup>
  <ServerList>
    <Server>
      <Name value="VALUE"/>
      ...
    </Server>
  </ServerList>
</ServerGroup>
```

3.4.2 Server Entry Parameters

You can use the following parameters in any **<Server>** entry, regardless of the **ServerType**.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|--------------------------|--------|---------|---|
| ConnectionMinPingTimeout | long | 20000 | Configures the minimum length of time (in milliseconds) to use as a timeout for a connected channel. |
| ConnectionPingTimeout | long | 60000 | Specifies the duration (in milliseconds) after which the Enterprise Message API terminates the connection if it does not receive communication or pings from the server. |
| CompressionThreshold | long | 30 | Sets the message size threshold (in bytes, the allowed value is 30-Integer.MAX_VALUE), above which all messages are compressed (thus individual messages might not be compressed). Different compression types have different behaviors and compression efficiency can vary depending on message size. |
| CompressionType | String | None | <p>Specifies the Enterprise Message API's preferred type of compression. Compression is negotiated between the client and server: if the server supports the preferred compression type, the server will compress data at that level.</p> <p>Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include:</p> <ul style="list-style-type: none"> • None • ZLib • LZ4 <p>NOTE: You can configure a server to force a particular compression type, regardless of client settings.</p> |
| DirectWrite | int | 0 | <p>Specifies whether to set the direct socket write flag when sending data on a channel.</p> <p>When the flag is set, every package is sent on the wire immediately on the submit call. If direct write is not set, the package might be placed into an internal queue which is later flushed onto the wire.</p> <p>Possible values are:</p> <ul style="list-style-type: none"> • 0: Send data without the direct socket write flag. • 1: Send data with the direct socket write flag. |
| GuaranteedOutputBuffers | long | 100 | <p>Specifies the number of guaranteed buffers (allocated at initialization time) available for use by each RsslChannel when writing data. Each buffer is created to contain maxFragmentSize bytes.</p> <p>For details on RsslChannel and maxFragmentSize, refer to the <i>Transport API Java Edition Developers Guide</i>.</p> |
| HighWaterMark | long | 6144 | Specifies the upper buffer-usage threshold for the channel. |
| InitializationTimeout | long | 60 | Specifies the time (in seconds) to wait for the successful initialization of a channel. |
| InterfaceName | String | "" | <p>Specifies a character representation of the IP address or hostname of the local network interface over which the Enterprise Message API sends and receives content.</p> <p>InterfaceName is for use in systems that have multiple network interface cards. If unspecified, the default network interface is used.</p> |
| Name | String | | Specifies the Server's name. |

Table 10: Universal <Server> Parameters

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|-----------------|--------|-------------|--|
| NumInputBuffers | long | 10 | Specifies the number of buffers used to read data. Buffers are sized according to maxFragmentSize . For details on RsslChannel and maxFragmentSize , refer to the <i>Transport API Java Edition Developers Guide</i> . |
| Port | String | 14002 | Specifies the port on the remote server to which the Enterprise Message API connects. |
| ServerType | String | RSSL_SOCKET | <p>Specifies the type of channel or connection used to connect to the server.</p> <p>Calling the host function can change this field. For details on this event, refer to Section 4.4.2.</p> <p>Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Available values include RSSL_SOCKET (0), RSSL_ENCRYPTED (1), or WEBSOCKET (7).</p> <p>RSSL_ENCRYPTED requires additional configuration provided via OmmlProviderConfig and programmatic configuration. For details, refer to Section 4.4.1.</p> <p>NOTE: Setting ServerType to RSSL_SOCKET or WEBSOCKET has the same behavior. An open WebSocket request from the client side notifies the server to update the socket to a WebSocket connection type. The application is responsible for accepting or rejecting traffic based on a protocol that it intends to support.</p> |
| SysRecvBufSize | long | 0 | Specifies the size (in bytes) of the system's receive buffer for this channel. For exact, effective values, refer to your operating system documentation. |
| SysSendBufSize | long | 0 | Specifies the size (in bytes) of the system's send buffer for this channel. For exact, effective values, refer to your operating system documentation. |
| TcpNodelay | int | 1 | <p>Specifies whether to use Nagle's algorithm when sending data. Available values are:</p> <ul style="list-style-type: none"> 0: Send data using Nagle's algorithm. 1: Send data without delay. |

Table 10: Universal <Server> Parameters (Continued)

3.4.3 Parameters for Use with ServerType WEBSOCKET

You can use the following parameter when **ServerType** is set to **WEBSOCKET**.

| PARAMETER NAME | TYPE | DEFAULT | DESCRIPTION |
|----------------|--------|----------------------------------|---|
| WsProtocols | String | tr_json2, rssl.rwf, rssl.json.v2 | Specifies a list of supported protocols in order of preference. Current protocols supported include rssl.json.v2 , rssl.rwf , and tr_json2 . |

Table 11: WEBSOCKET ServerType Parameter

3.5 Dictionary Group

The **DictionaryGroup** contains a **DictionaryList**, which contains one or more **Dictionary** components (each uniquely identified by a **<Name .../>** entry). Each **Dictionary** component defines parameters relating to how the dictionary is accessed.

3.5.1 Generic XML Schema for DictionaryGroup

The top-level XML schema for **DictionaryGroup** is as follows:

```
<DictionaryGroup>
  <DictionaryList>
    <Dictionary>
      <Name value="..." />
      ...
    </Dictionary>
  </DictionaryList>
</DictionaryGroup>
```

3.5.2 Dictionary Entry Parameters

Use the following parameters when configuring a **Dictionary** entry in the Enterprise Message API.

| PARAMETER NAME | TYPE | DEFAULT | NOTES |
|----------------------------|--------|-------------------|--|
| DictionaryType | String | ChannelDictionary | Specifies the dictionary loading mode. Use strings with Enterprise Message API's programmatic configuration (for further details, refer to Section 4.5). Possible values are: <ul style="list-style-type: none"> FileDictionary (0): The Enterprise Message API loads the dictionaries from the files specified in the parameters RdmFieldDictionaryFileName and EnumTypeDefFileName. ChannelDictionary (1): The Enterprise Message API downloads dictionaries by requesting the dictionaries from the upstream provider. |
| EnumTypeDefFileName | String | | Sets the location of the EnumTypeDef file. |
| EnumTypeDefItemName | String | RWFEnum | Sets the name of the EnumTypeDef item specified in the source directory InfoFilter.DictionariesProvided , and InfoFilter.DictionariesUsed elements. |
| Name | String | | Sets a unique name for a Dictionary component in the DictionaryList . |
| RdmFieldDictionaryFileName | String | | Sets the location of the RdmFieldDictionary . |
| RdmFieldDictionaryItemName | String | RWFFld | Sets the name of the RdmFieldDictionary item specified in the source directory InfoFilter.DictionariesProvided , and InfoFilter.DictionariesUsed elements. |

Table 12: Dictionary Group Parameters

3.6 Directory Group

The **DirectoryGroup** contains a **DirectoryList**, which contains one or more **Directory** components (each uniquely identified by a **<Name .../>** entry). Each **Directory** component defines a list of **Service** components (which in turn define parameters that relate to the Service **InfoFilter** and **StateFilter**).

3.6.1 Generic XML Schema for Directory Entry

The top-level XML schema for **DirectoryGroup** is as follows:

```
<DirectoryGroup>
  <DefaultDirectory value="..." />
  <DirectoryList>
    <Directory>
      <Name value="..." />
      <Service>
        <Name value="..." />
        <InfoFilter>
          ...
        </InfoFilter>
        <StateFilter>
          ...
        </StateFilter>
      </Service>
    </Directory>
    ...
  </DirectoryList>
</DirectoryGroup>
```

3.6.2 Setting Default Directory

If you do not specify a **DefaultDirectory**, then the Enterprise Message API uses the first **Directory** component in the **DirectoryGroup**. However, you can specify a default directory by including the following parameter on a unique line inside **DirectoryGroup** but outside **DirectoryList**.

```
<DefaultDirectory value="VALUE" />
```

3.6.3 Configuring a Directory in a DirectoryGroup

To configure a **Directory** component, add the following parameters (as appropriate) to the target directory in the XML Schema, each on a separate line:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-----------|----------------|---------|---|
| Name | String | N/A | Specifies the name of this Directory component. Name is required when creating a Directory component. You can use any value for Name . |
| Service | Component Name | N/A | Specifies InfoFilter and StateFilter values for the given Service . NOTE: A Directory may contain several Service components. |

Table 13: Directory Entry Parameters

3.6.4 Service Entry Parameters

The Service Entry resembles the RDM's Source Directory Domain payload. For further details, refer to the *Enterprise Message API Java Edition RDM Usage Guide*. The Enterprise Message API supports only the RDM entries **InfoFilter** and **StateFilter**. Use the following parameters when configuring a Service in the Enterprise Message API:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-------------|----------------|---------|--|
| Name | String | N/A | Specifies the name of this Service component. You can use any value for Name . |
| InfoFilter | Component Name | N/A | Specifies InfoFilter values for the given Service . InfoFilter values set a filter on the types of information that the Enterprise Message API sends out. |
| StateFilter | Component Name | N/A | Specifies StateFilter values for the given Service . The Enterprise Message API sends StateFilter values to describe the service's state. |

Table 14: Service Entry Parameters

3.6.5 InfoFilter Entry Parameters

The Enterprise Message API uses the following **InfoFilter** parameters to set filters on the types of information it sends over its services (as specified in the **EmaConfig.xml**).

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-----------|--------|---------|---|
| ServiceId | int | N/A | Specifies the Service 's unique identifier. Available values include 0 - 65535. |
| Vendor | String | N/A | Specifies the name of the vendor that provides the service. |
| IsSource | int | 0 | Specifies whether the source of data sent on this service is its original publisher: <ul style="list-style-type: none"> 1: The service's data is provided directly by an original publisher 0: The service's data is a consolidation of multiple sources into a single service. |

Table 15: Source Directory Info Parameters

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------|----------------|---------------------------------|---|
| Capabilities | Component Name | N/A | A component that includes CapabilitiesEntry parameters, which define the message domain types that can be requested from the service. For details on the parameter used in this section, refer to Section 3.6.5.1. |
| ItemList | String | N/A | Specifies the name of the SymbolList that includes all items provided by this service. |
| DictionariesProvided | Component Name | N/A | A component that includes DictionariesProvidedEntry parameters, which define the dictionaries that the provider makes available. When specifying a dictionary, use the Dictionary's component name whose *ItemName entries are used in this Service's RDM DictionariesProvided entry. For details on the parameter used in this section, refer to Section 3.6.5.2. |
| AcceptingConsumerStatus | int | 1 | Indicates whether a service can accept and process messages related to Source Mirroring. <ul style="list-style-type: none"> 0: The provider does not accept consumer status 1: The provider accept consumer status |
| DictionariesUsed | Component Name | N/A | A component that includes DictionariesUsedEntry parameters, which define the dictionaries that the provider uses. When specifying a dictionary, use the Dictionary's component name whose *ItemName entries are used in this Service's RDM DictionariesUsed entry. For details on the parameter used in this section, refer to Section 3.6.5.3. |
| QoS | Component Name | Includes a single QoSEntry | A component that includes QoSEntry sections, with each QoSEntry section defining a QoS Timeliness and Rate supported by this Service. For details on the parameter used in this section, refer to Section 3.6.5.4. |
| SupportsQoSRange | int | 0 | Indicates whether the provider supports a QoS range when requesting an item. <ul style="list-style-type: none"> 0: The provider does not support a QoS Range. 1: The provider supports a QoS Range. For further details on using QoS ranges, refer to the <i>RDM Java Edition Usage Guide</i> . |
| SupportsOutOfBandSnapshots | int | For non-interactive provider: 0 | Indicates whether the provider supports Snapshot requests after the OpenLimit has been reached: <ul style="list-style-type: none"> 0: The provider does not support snapshot requests. 1: The providers supports snapshot requests. For details on OpenLimit , refer to the <i>RDM Java Edition Usage Guide</i> . |

Table 15: Source Directory Info Parameters (Continued)

3.6.5.1 CapabilitiesEntry Parameter

Use the **CapabilitiesEntry** parameter to configure the message domain type supported by the **Service** component:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-------------------|---------------|---------|---|
| CapabilitiesEntry | int or String | N/A | Specifies the message domain type supported by the Service component. Accepted names are listed in the EmaRdm interface. |
| | | | NOTE: You can set CapabilitiesEntry to be an RDM domain number or name (e.g. 6 or MMT_MARKET_PRICE). |

Table 16: CapabilitiesEntry Parameter

3.6.5.2 DictionariesProvided Entry Parameter

Use the **DictionariesProvidedEntry** parameter to configure the dictionaries provided for the **Service's InfoFilter**:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|---------------------------|--------|---|---|
| DictionariesProvidedEntry | String | RWFFId for RdmFieldDictionaryItemName RWFFEnum for enumTypeDefItemName | Specifies the name of a Dictionary component from the DictionaryGroup section whose RdmFieldDictionaryItemName and enumTypeDefItemName parameters are used in this Service's RDM DictionariesProvided entry. |

Table 17: DictionariesProvided Parameter

3.6.5.3 DictionariesUsed Entry Parameter

Use the **DictionariesUsedEntry** parameter to configure the types of dictionaries used by the **Service's InfoFilter**:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-----------------------|--------|---|--|
| DictionariesUsedEntry | String | RWFFId for RdmFieldDictionaryItemName RWFFEnum for enumTypeDefItemName | Specifies the name of a Dictionary component from the DictionaryGroup section whose RdmFieldDictionaryItemName and enumTypeDefItemName are used in this Service's RDM DictionariesUsed entry. |

Table 18: DictionariesUsedEntry Parameter

3.6.5.4 QoSEntry Section and Associated Parameters

Use a **QoSEntry** section to configure a specific QoS supported by the **Service's InfoFilter**. You can include multiple QoSEntry sections in a parent **QoS** section.

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|------------|---------------|----------------------|--|
| QoSEntry | | N/A | QoSEntry is the name of a section that contains parameters specifying the Timeliness and Rate parameters for a given QoS. You can use multiple QoSEntry sections for a Service's InfoFilter . |
| Timeliness | int or String | Timeliness::Realtime | Specifies the QoS timeliness, which describes the age of the data (e.g., real time). NOTE: You can use numbers or names. Accepted names are listed in the OmmQos.Timeliness class. |
| Rate | int or String | Rate:TickByTick | Specifies the QoS rate, which is the rate of change for data sent over the Service . NOTE: You can use numbers or names. Accepted names are listed in the OmmQos.Rate class. |

Table 19: QoSEntry Section and Associated Parameters

3.6.6 StateFilter Entry Parameters

Use the following parameters to configure the **Service's StateFilter** (as specified in the **EmaConfig.xml**), which communicates the service's state.

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-------------------|------|---------------------------------|--|
| ServiceState | int | N/A | Specifies whether the service is up or down: <ul style="list-style-type: none"> 0: Service is down 1: Service is up |
| AcceptingRequests | int | For non-interactive provider: 0 | Specifies whether the service accepts request messages: <ul style="list-style-type: none"> 0: The provider does not accept request messages. 1: The provider accepts request messages. |
| Status | | Open / Ok / None / "" | Specifies a change in status to apply to all items provided by this service. The status only applies to items that received an OPEN/OK in a refresh or status message. |

Table 20: StateFilter Parameters

3.6.7 Status Entry Parameters

Use the following parameters when configuring the **Service's StateFilter**:

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|-------------|--------|-------------------|--|
| StreamState | String | StreamState::Open | Specifies the state of the item stream. |
| | | | NOTE: Acceptable StreamState values are listed in the OmmState.StreamState class. |
| DataState | String | DataState::Ok | Specifies the state of the item data. |
| | | | NOTE: Acceptable DataState values are listed in the OmmState.DataState class. |
| StatusCode | String | StatusCode::None | Specifies the item status code. |
| | | | NOTE: Codes and their meanings are listed in the OmmState.StatusCode class. |
| StatusText | String | "" | Specific StatusText regarding the current data and stream state. Typically used for informational purposes. StatusText has an encoded text with a maximum allowed length of 32,767 bytes. |

Table 21: Status Entry Parameters

3.7 Global Configuration

GlobalConfig contains parameters that relate to the entire application.

3.7.1 Generic XML Schema for GlobalConfig

The top-level XML schema for **GlobalConfig** is as follows:

```
<GlobalConfig>
...
</GlobalConfig>
```

3.7.2 Parameters

Enterprise Message API uses the following global parameters. All parameters are optional.

| PARAMETER | TYPE | DEFAULT | DESCRIPTION |
|----------------------------------|------|---------|--|
| ReactorMsgEventPoolLimit | int | -1 | Specifies the maximum number of events in the ReactorMsgEvent pool. When the limit is reached, events will be created outside of the pool and garbage collected later. If this value is negative (such as the default setting), then amount of events is unlimited. |
| ReactorChannelEventPoolLimit | int | -1 | Specifies the maximum number of events in the ReactorChannelEvent pool. When the limit is reached, events are created outside of the pool and garbage collected later. If this value is negative (such as the default setting), then the amount of events is unlimited. |
| WorkerEventPoolLimit | int | -1 | Specifies the maximum number of events in the WorkerEvent pool. When the limit is reached, events are created outside of the pool and garbage collected later. If the value is negative (such as the default setting), then the amount of events is unlimited. |
| TunnelStreamMsgEventPoolLimit | int | -1 | Specifies the maximum number of events in the TunnelStreamMsgEvent pool. When the limit is reached, events are created outside of the pool and will be garbage collected later. If value is negative (such as the default setting), then the amount of events is unlimited. |
| TunnelStreamStatusEventPoolLimit | int | -1 | Specifies the maximum number of events in the TunnelStreamStatusEvent pool. When the limit is reached, events are created outside of the pool and garbage collected later. If the value is negative, then the amount of events is unlimited. |

Table 22: Global Configuration Parameters

4 EMA Configuration Processing

4.1 Overview and Configuration Precedence

The Enterprise Message API configuration is determined by hard-coded behaviors, customized behaviors as specified in a configuration file (i.e., **EmaConfig.xml**), programmatic changes, and other internal processing. All of these vectors affect Enterprise Message API's configuration as used by application components. The Enterprise Message API merges configuration parameters specified from all vectors with the following precedence: Function calls, Programmatic Configuration, File Configuration (such as **EmaConfig.xml**), and finally the default configuration (i.e., if parameters are specified in both function calls and the programmatic configuration, the function call configuration takes precedence).

4.2 Default Configuration

4.2.1 Default Consumer Configuration

Each Enterprise Message API consumer-type application must eventually instantiate an **OmmConsumer** object. Constructors for **OmmConsumer** require a **OmmConsumerConfig** object. The **OmmConsumerConfig** constructor can read and process an optional XML file, which applications can use to modify Enterprise Message API's default consumer behavior. By default this file is named **EmaConfig.xml** and stored in the working directory. For details on using non-default names and directories for your XML configuration file, refer to Section 4.3.1.2.

The Enterprise Message API provides a hard-coded configuration for use whenever an **OmmConsumerConfig** object is instantiated without a configuration file (such as **EmaConfig.xml**) in the run-time environment. The resulting configuration is created by taking the defaults from the various configuration groups. For example, the default (hard-coded) behavior for a **Channel** adheres to the following configuration:

- **ChannelType** value="RSSL_SOCKET"
- **CompressionType** value="None"
- **TcpNoDelay** value="1"
- **Host** value="localhost"
- **Port** value="14002"

Note that unlike the Enterprise Message API's default behavior of choosing the first **Consumer** component in the **ConsumerList**, Enterprise Message API applications will not choose the first **Channel** or **Dictionary** in their respective lists. Instead, if an application wants to use a specific channel or dictionary configuration, the application must explicitly configure it in the appropriate **Consumer** section of the XML file.

For specifics on Enterprise Message API's default configuration, refer to Section 2.3.

4.2.2 Default Provider Configurations

Each Enterprise Message API provider-type application must eventually instantiate an **OmmProvider** object. Constructors for **OmmProvider** require a **OmmProviderConfig** object. The **OmmProviderConfig** constructor can read and process an optional XML file, which applications can use to modify the Enterprise Message API's default provider behavior. By default this file is named **EmaConfig.xml** and stored in the working directory. For details on using non-default names and directories for your XML configuration file, refer to Section 4.3.1.2.

The Enterprise Message API provides a hard-coded configuration for use whenever an **OmmProviderConfig** object is instantiated without an **EmaConfig.xml** file in the run-time environment. The resulting Enterprise Message API configuration is created by taking the defaults from the various configuration groups.

4.2.2.1 Example: Default Channel Behavior (NiProvider)

The default (hard-coded) behavior for a **Channel** adheres to the following configuration:

- **ChannelType** value="RSSL_SOCKET"
- **CompressionType** value="None"
- **TcpNoDelay** value="1"
- **Host** value="localhost"
- **Port** value="14003"

Note that unlike the Enterprise Message API's default behavior of choosing the first **NiProvider** component in the **NiProviderList**, Enterprise Message API applications will not choose the first or **Channel** in their respective lists. Instead, if an application wants to use a specific channel or dictionary configuration, the application must explicitly configure it in the appropriate **NiProvider** section of the XML file.

4.2.2.2 Example: Default Server Behavior (IProvider)

The default (hard-coded) behavior for a **Server** adheres to the following configuration:

- **ServerType** value="RSSL_SOCKET"
- **CompressionType** value="None"
- **TcpNoDelay** value="1"
- **Port** value="14002"

Note that unlike the Enterprise Message API's default behavior of choosing the first **IProvider** component in the **IProviderList**, Enterprise Message API applications will not choose the first or **Server** in their respective lists. Instead, if an application wants to use a specific server or dictionary configuration, the application must explicitly configure it in the appropriate **IProvider** section of the XML file.

4.3 Processing EMA's XML Configuration File

The Refinitiv Real-Time SDK package installs a default configuration file named **EmaConfig.xml** into the Enterprise Message API's working directory. By default, the Enterprise Message API looks for a configuration file with this name in the working directory. If you want to use a different name for your configuration file, and/or store the file in a directory other than the working directory, you must specify this filename and/or directory in your configuration object. For further details on using the configuration object, how it functions as regards paths and filenames, and how the Enterprise Message API determines its configuration, refer to Section 4.3.1.

Except for the parameters **DefaultConsumer** and **DefaultNiProvider**, you must wrap all other elements defined in the Enterprise Message API's configuration file in a component definition (i.e., **Consumer**, **NiProvider**, **Channel**, **Directory**, or **Dictionary**) otherwise the Enterprise Message API ignores the element. This section includes some examples that illustrate this requirement.

4.3.1 Reading the Configuration File

NOTE: The following section uses Consumer objects (i.e., **OmmConsumer** and **OmmConsumerConfig**) to illustrate how the Enterprise Message API checks for a configuration file, and if one exists, how the Enterprise Message API starts to process it. For details on interactive and non-interactive providers (instead of consumers) and their OmmProvider-type objects, refer to the *Enterprise Message API Java Developers Guide*.

The **OmmConsumer** constructor expects an **OmmConsumerConfig** object. By default, **OmmConsumerConfig** searches its working directory for a configuration file by the name of **EmaConfig.xml**. However, if you store your configuration file elsewhere on the system, or use a custom filename, you can include an argument with the configuration object to specify the alternate path and/or name of your configuration file.

4.3.1.1 Using EmaConfig.xml in CLASSPATH or in the Working Directory

If **OmmConsumerConfig** lacks an argument, the application attempts to open a configuration file named **EmaConfig.xml**, which must be located in either the **CLASSPATH** root directories or in the current working directory. By precedence, the Enterprise Message API uses the **EmaConfig.xml** from the **CLASSPATH** root directories; if one does not exist in the **CLASSPATH**, then the Enterprise Message API uses the **EmaConfig.xml** from the current working directory. If the Enterprise Message API does not find an **EmaConfig.xml**, the application uses the default configuration. Additionally, if the **EmaConfig.xml** file is empty or contains malformed XML, the application uses the default configuration. For details on the default configuration, refer to Section 4.2.

For example, to use an **EmaConfig.xml** stored in CLASSPATH or in the working directory, have the application create an **OmmConsumerConfig** object (for details on this object, refer to the *Enterprise Message API Java Developers Guide*) and pass it to the **OmmConsumer** object as follows:

```
OmmConsumerConfig config = EmaFactory.createOmmConsumerConfig();

consumer = EmaFactory.createOmmConsumer(config);
```

For complete details, you can refer to the example *example100__MarketPrice__Streaming* included with the Refinitiv Real-Time SDK.

4.3.1.2 Using a Custom Filename and/or Directory

If you include a path with `OmmConsumerConfig`, the application creates a filename from the argument and attempts to open a file with that name, as follows:

- If the argument represents only a directory, the Enterprise Message API appends **EmaConfig.xml** to the argument and verifies whether **EmaConfig.xml** exists in the specified directory.
- If the argument represents a directory and filename, the Enterprise Message API verifies whether the specified file exists.
- If the specified file does not exist, the application throws an `IceException`, which indicates the specified path and the current working directory.
- If the argument represents neither a file nor a directory, an `IceException` is thrown.

At this point, the application attempts to create an XML configuration from the filename. If the attempt fails, the application throws an `IceException`.

If you want to specify a custom path and filename, have the application create an `OmmConsumerConfig` object with the path and filename in the argument (for details on this object, refer to the *Enterprise Message API Java Developers Guide*) and pass it to the `OmmConsumer` object as follows (where **PATH** is the alternate path and/or filename you want to use for your configuration file):

```
OmmConsumerConfig config = EmaFactory.createOmmConsumerConfig(PATH);

consumer = EmaFactory.createOmmConsumer(config);
```

For an example of how to specify a custom configuration file name, refer to *Example 111* (**example111__MarketPrice__UserSpecifiedFileConfig**) included with the package.

4.3.2 Use of the Correct Order in the XML Schema

In the following configuration file snippet (only those parts needed for the example are included), the application creates a consumer with a **Name of Consumer_1**.

```
<ConsumerGroup>
  <ConsumerList>
    <Consumer>
      <Name value="Consumer_1"/>
    </Consumer>
  </ConsumerList>
</ConsumerGroup>
```

Now assume that the following was not included in the XML configuration:

```
<Directory>
  <Name value="Directory_1"/>
```

In this case, the Enterprise Message API application relies on its hard-coded behavior.

However, if the snippet is configured in either of the following configurations, the Enterprise Message API application reverts to its default behaviors because the parameters are not in the correct order (i.e., the **Name** parameter needs to be contained in a **Directory** component entry):

- Configuration 1:

```
<DirectoryGroup>
  <Name value="Name"/>
  <DirectoryList>
    ...
```

- Configuration 2:

```
<DirectoryGroup>
  <DirectoryList>
    <Name value="Name"/>
    <Directory>
      ...
```

4.3.3 Processing the Consumer “Name”

The Enterprise Message API is hard-coded to use a default consumer of **EmaConsumer**. However, you can change this by using the configuration file (e.g., **EmaConfig.xml**). When you use the XML file, the default **Consumer Name** is either specified by the **DefaultConsumer** element, or if this parameter is not set, then the Enterprise Message API application will default to the name of the first Consumer component.

- If **DefaultConsumer** uses an invalid name (i.e., no **Consumer** components in the XML file use that name), the Enterprise Message API throws an exception indicating that **DefaultConsumer** is invalid.
- If the configuration file has no **Consumer** components, the Enterprise Message API application uses **EmaConsumer**.

4.3.4 Processing the Provider “Name”

The Enterprise Message API is hard-coded to use a default non-interactive provider of **EmaProvider**. However, you can change this by using the configuration file (e.g., **EmaConfig.xml**). When you use the XML file, the default **Provider Name** is either specified by the **DefaultProvider** element, or if this parameter is not set, then the Enterprise Message API application will default to the name of the first non-interactive provider component.

- If **DefaultProvider** uses an invalid name (i.e., no **Provider** components in the XML file use that name), the Enterprise Message API throws an exception indicating that **DefaultProvider** is invalid.
- If the **EmaConfig.xml** has no **Provider** components, the Enterprise Message API application uses **EmaProvider**.

4.4 Configuring EMA Using Function Calls

From an application standpoint, instantiating `OmmConsumerConfig` and `OmmNiProviderConfig` objects creates the initial configuration from the Enterprise Message API's XML configuration file (if one exists). Certain variables can then be altered via function calls on the `OmmConsumerConfig` and `OmmProviderConfig` objects.

NOTE: Function calls override any settings in a configuration XML file.

4.4.1 EMA Configuration Method Calls

4.4.1.1 OmmConsumerConfig Class Method Calls

You can use the following method calls in an Enterprise Message API consumer application:

| METHOD | DESCRIPTION |
|---|---|
| <code>addAdminMsg(ReqMsg)</code> | Populates part of or all of the login request message, directory request message, or dictionary request message according to the specification discussed in the <i>Enterprise Message API Reuters Domain Models (RDM) Usage Guide</i> specific to the programming language you use. |
| <code>applicationId(String)</code> | Sets the applicationId variable. applicationId has no default value. |
| <code>clear()</code> | Clears existing content from the <code>OmmConsumerConfig</code> object. |
| <code>config(const Data&)</code> | Passes in the consumer's programmatic configuration. |
| <code>consumerName(String)</code> | Sets the consumer name, which is used to select a specific consumer as defined in the Enterprise Message API's configuration. If a consumer does not exist with that name, the application throws an exception. |
| <code>clientId(String)</code> | Sets the clientId variable. clientId has no default value. clientId specifies a unique ID for application making the request to the RDP token service. |
| <code>host(String)</code> | Sets the host and port parameters. For details, refer to Section 4.4.2. |
| <code>operationModel(OperationModel)</code> | Sets the operation model to either <code>OperationModel.API_DISPATCH</code> (which is the default) or <code>OperationModel.USER_DISPATCH</code> . |
| <code>password(String)</code> | Sets the password variable. password has no default value. |
| <code>position(String)</code> | Sets the position variable. position has no default value. |
| <code>username(String)</code> | Sets the username variable. If username is not set, the application extracts a username from the run-time environment. |

Table 23: OmmConsumerConfig Class Method Calls

4.4.1.2 OmmProviderConfig Class Method Calls

You can use the following method calls in an Enterprise Message API **Provider** application. For further details on variables, refer to the *Enterprise Message API Java RDM Usage Guide*. Certain method calls can only be used with a specific provider type (e.g., `addAdminMsg(const ReqMsg&)` can only be used with an **NiProvider**). The parameter's description will mention any provider-type restrictions.

| METHOD | DESCRIPTION |
|--|--|
| <code>addAdminMsg(ReqMsg)</code> | Used only with NiProvider . Populates part of or all of the login request message according to the specification discussed in the <i>Enterprise Message API Java RDM Usage Guide</i> . |
| <code>addAdminMsg(RefreshMsg)</code> | Populates part of or all of the initial directory refresh message according to the specification discussed in the <i>Enterprise Message API Java RDM Usage Guide</i> . |
| <code>adminControlDirectory(int)</code> | Specifies whether the API or the user controls the sending of Directory refresh messages. Available values include: <ul style="list-style-type: none"> <code>OmmProviderConfig.AdminControl.API_CONTROL</code> (which is the default) <code>OmmProviderConfig.AdminControl.USER_CONTROL</code> For details on control models, refer to OmmProviderConfig.h . |
| <code>applicationId(String)</code> | Used only with NiProvider . Sets the <i>applicationId</i> variable. <i>applicationId</i> has no default value. |
| <code>clear()</code> | Clears existing content from the OmmProviderConfig object. |
| <code>config(Data)</code> | Passes in the provider's programmatic configuration. |
| <code>host(String)</code> | Used only with NiProvider . Sets the host and port parameters. For details, refer to Section 4.4.2. |
| <code>instanceId(String)</code> | Used only with NiProvider . Sets the <i>instanceId</i> variable. <i>instanceId</i> has no default value. |
| <code>keyManagerAlgorithm(String)</code> | Used only with IProvider . Specifies the key manager algorithm for the configured server. This defaults to SunX509 . |
| <code>keystoreFile(String)</code> | Used only with IProvider . Specifies the keystore file for an encrypted connection, containing the server private key and certificate. If <code>keystoreFile(String)</code> is not set, by default, the application loads the JVM's default keystore set with javax.net.ssl.keystore . |
| <code>keystorePasswd(String)</code> | Used only with IProvider . Specifies the keystore password for the configured keystore. If <code>keystorePasswd(String)</code> is not set, by default, the application loads the JVM's default keystore set with javax.net.ssl.keystorepassword . |
| <code>keystoreType(String)</code> | Used only with IProvider . Specifies the keystore type for the configured keystore. This defaults to JKS . |
| <code>operationModel(int)</code> | Specifies whether the API or the user controls the thread (i.e., the operation model). Available values include: <ul style="list-style-type: none"> <code>OmmProviderConfig.OperationModel.API_DISPATCH</code> (which is the default) <code>OmmProviderConfig.OperationModel.USER_DISPATCH</code> For details on operation models, refer to OmmProviderConfig.h . |

Table 24: OmmProviderConfig Class Method Calls

| METHOD | DESCRIPTION |
|---------------------------------|---|
| password(String) | Used only with NiProvider . Sets the password variable. password has no default value. |
| port() | Sets the port parameters. |
| position(String) | Used only with NiProvider . Sets the position variable. position has no default value. |
| providerName(String) | Sets the provider's name, which is used to select a specific provider as defined in the Enterprise Message API's configuration. If a provider does not exist with that name, the application throws an exception. |
| securityProtocol(String) | Used only with IProvider . Specifies which TLS protocol to use with this server. This defaults to TLSv1.2 . |
| securityProvider(String) | Used only with IProvider . Specifies the security provider type for the configured server. This defaults to SunJSSE . |
| trustManagerAlgorithm(String) | Used only with IProvider . Specifies the trust manager algorithm for the configured server. This defaults to PKIX . |
| username(String) | Used only with NiProvider . Sets the username variable. If username is not set, the application extracts a username from the run-time environment. |

Table 24: **OmmProviderConfig** Class Method Calls (Continued)

4.4.2 Using the **host()** Function: How Host and Port Parameters are Processed

Host and **Port** parameters both have global default values. Thus, if either an **OmmConsumerConfig** or **OmmNiProviderConfig** object exists, its **Host** and **Port** will always have values (either the default value or some other value as specified in a configuration XML file such as **EmaConfig.xml**).

- The default **Host:Port** value for **OmmConsumerConfig** is **localhost:14002**.
- The default **Host:Port** value for **OmmNiProviderConfig** is **localhost:14003**.

If needed, you can have the application reset both host and port values by calling the **host(String)** method on the object using the syntax: **HostValue:PortValue**.

NOTE: Calling the **host()** function sets **channelType** (refer to Section 3.3.2) to **RSSL_SOCKET**, regardless of how it was previously configured.

Host and **Port** values observe the following rules when updating due to the **host(String)** method:

- If the host parameter is missing or empty, then host and port reset to their global default values.
- If the host parameter is set to the string “:”, then host and port reset to their global default values.
- If the host parameter is a string (not containing a :), then host is set to that string and port resets to its default value.
- If the parameter begins with a : and is followed by some text, then host is set to its global default value and port is set to that text.
- If the parameter is **HostValue:PortValue**, where both **HostValue** and **PortValue** have values, then host is set to **HostValue** and port is set to **PortValue**.

4.5 Programmatic Configuration

In addition to changing the Enterprise Message API's configuration via an XML configuration file (e.g., **EmaConfig.xml**) or function calls, you can programmatically change the API's behavior via an OMM data structure.

4.5.1 OMM Data Structure

Programmatic configuration of the Enterprise Message API provides a way of configuring all parameters using an OMM data structure, which is divided into four tiers:

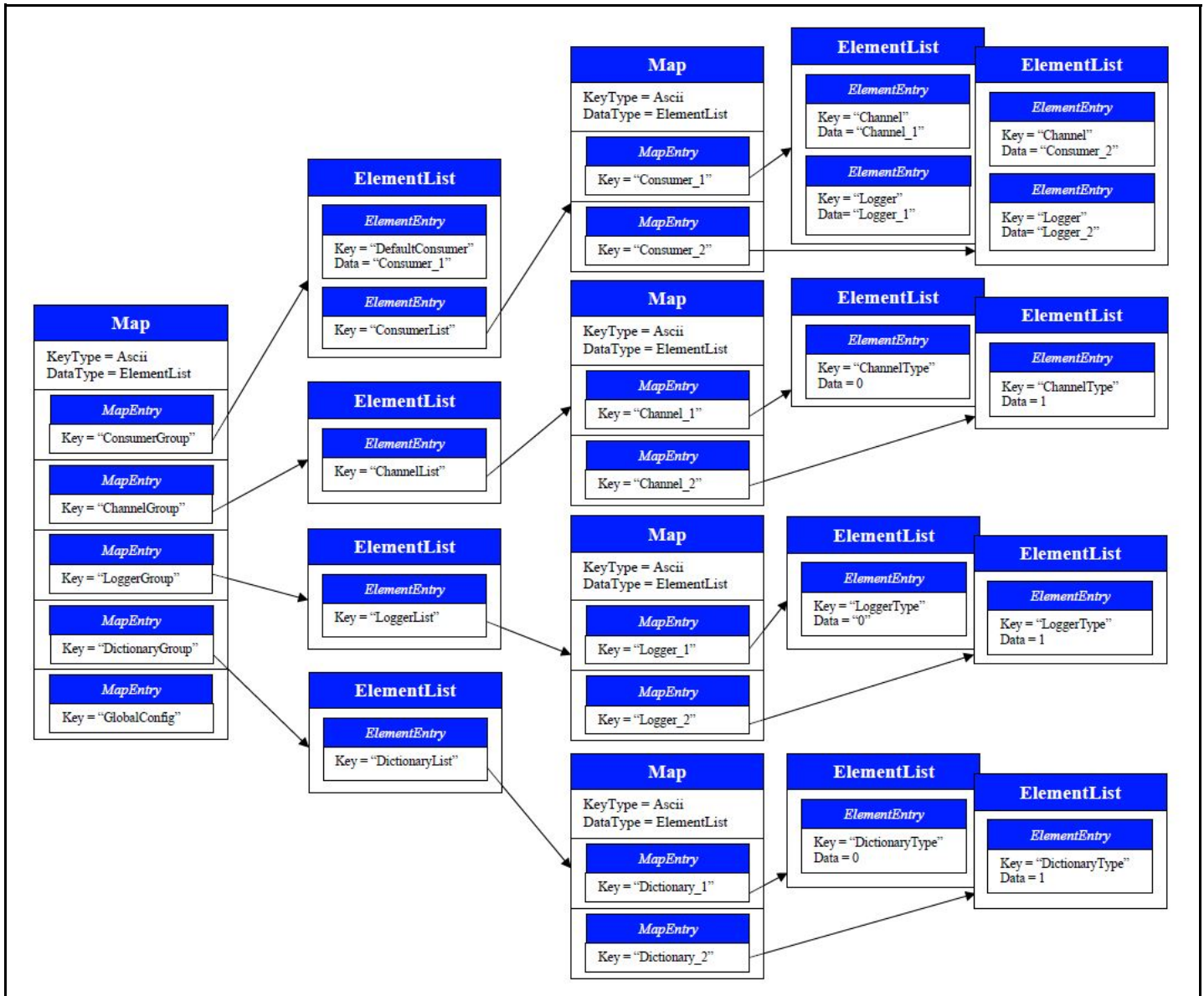
- The 1st tier lists the Enterprise Message API's Consumer, IProvider, NiProvider, Channel, Logger, Directory, and Dictionary components; each of which has its own list in the 2nd tier.
- The 2nd tier includes each component's list and the default consumers and providers for use when loading configuration parameters.
- The 3rd tier defines individual names for these components, which then have their own configuration parameters in 4th tier.
- The 4th tier defines configuration parameters that are assigned to specific components.

4.5.2 Creating a Programmatic Configuration for a Consumer

NOTE: When encoding OMM types, you must follow the OMM data structure and configuration parameter types listed in this document.

► **To programmatically configure an Enterprise Message API consumer:**

1. Create a map with the following hierarchy to configure Enterprise Message API configuration parameters:



2. Call the **config** method on an **OmmConsumerConfig** object, and pass the Map (which represents the programmatic OMM structure) as a parameter to the **config** method.

You can pass in multiple maps, each programmatic configuration being applied to create the application's active configuration during instantiation of the **OmmConsumer** or **OmmProvider**.

4.5.3 Example: Programmatic Configuration of the Consumer

The following example illustrates programmatically configuring a consumer:

```
Map innerMap = EmaFactory.createMap();
Map configMap = EmaFactory.createMap();
ElementList elementList = EmaFactory.createElementList();
ElementList innerElementList = EmaFactory.createElementList();

innerElementList.add(EmaFactory.createElementEntry().intValue("ReactorMsgEventPoolLimit", 2000));
innerElementList.add(EmaFactory.createElementEntry().intValue("ReactorChannelEventPoolLimit", 1500));
innerElementList.add(EmaFactory.createElementEntry().intValue("WorkerEventPoolLimit", 1000));
innerElementList.add(EmaFactory.createElementEntry().intValue("TunnelStreamMsgEventPoolLimit",
    2500));
innerElementList.add(EmaFactory.createElementEntry().intValue("TunnelStreamStatusEventPoolLimit",
    3000));
configMap.add(EmaFactory.createMapEntry().keyAscii( "GlobalConfig", MapEntry.MapAction.ADD,
    innerElementList ));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().ascii("DefaultConsumer", "Consumer_1"));
innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelSet", "Channel_1, Channel_2"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Dictionary", "Dictionary_1"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Consumer_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map( "ConsumerList", innerMap ));
innerMap.clear();
configMap.add(EmaFactory.createMapEntry().keyAscii( "ConsumerGroup", MapEntry.MapAction.ADD,
    elementList ));
elementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelType",
    "ChannelType::RSSL_SOCKET"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Host", "localhost"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Port", "14002"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Channel_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("ChannelType",
    "ChannelType::RSSL_SOCKET"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Host", "121.1.1.100"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Port", "14008"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Channel_2", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map( "ChannelList", innerMap ));
```

```

innerMap.clear();

configMap.add(EmaFactory.createMapEntry().keyAscii( "ChannelGroup", MapEntry.MapAction.ADD,
    elementList ));
elementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("DictionaryType",
    "DictionaryType::ChannelDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("RdmFieldDictionaryFileName", "./
    RDMFieldDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("EnumTypeDefFileName", "./enumtype.def"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Dictionary_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map( "DictionaryList", innerMap ));
configMap.add(EmaFactory.createMapEntry().keyAscii( "DictionaryGroup", MapEntry.MapAction.ADD,
    elementList ));
elementList.clear();

...

consumer = EmaFactory.createOmmConsumer(EmaFactory.createOmmConsumerConfig().config(configMap));

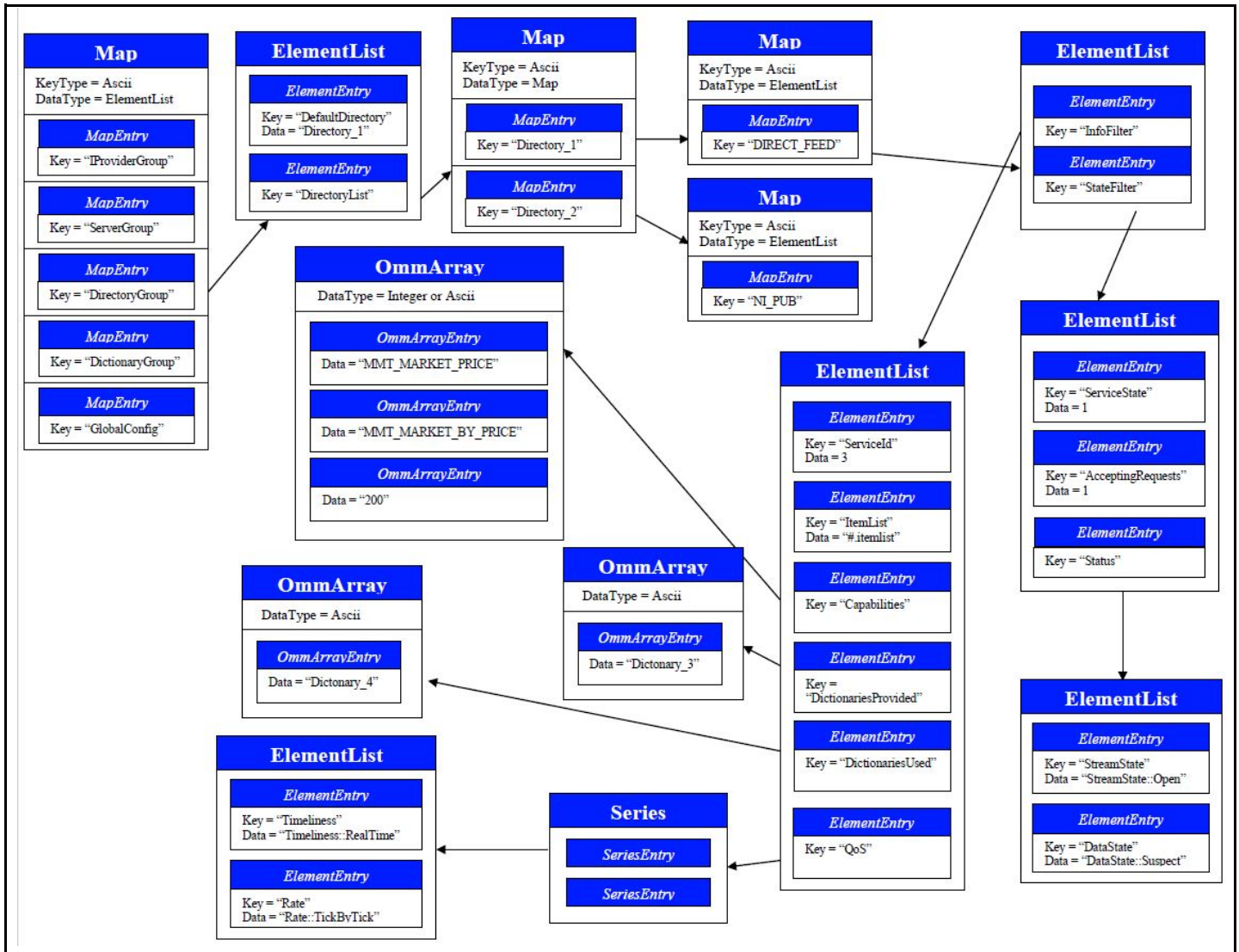
```

4.5.4 Creating a Programmatic Configuration for a Provider

NOTE: When encoding OMM types, you must follow the OMM data structure and configuration parameter types listed in this document.

► To programmatically configure an Enterprise Message API Provider:

1. To configure an Enterprise Message API directory's configuration parameters, create a map with the following hierarchy:



2. Call the **config** method on an **OmmProviderConfig** object, and pass the Map (which represents the programmatic OMM structure) as a parameter to the **config** method.

You can pass in multiple maps, each programmatic configuration being applied to create the application's active configuration during instantiation of the **OmmConsumer** or **OmmProvider**.

NOTE: You must set **adminControlDirectory** and **adminControlDictionary** to their default settings (**ApiControlEnum**) when programmatically configuring:

- A Directory Refresh message published by a *Provider*, or
- A Dictionary Refresh message published by an *IPProvider*

4.5.5 Example: Programmatic Configuration of a Provider

The following example illustrates programmatically configuring a *Provider*:

```
Map outermostMap = EmaFactory.createMap();
Map innerMap = EmaFactory.createMap();
ElementList elementList = EmaFactory.createElementList();
ElementList innerElementList = EmaFactory.createElementList();

innerElementList.add(EmaFactory.createElementEntry().intValue("ReactorMsgEventPoolLimit", 2000));
innerElementList.add(EmaFactory.createElementEntry().intValue("ReactorChannelEventPoolLimit", 1500));
innerElementList.add(EmaFactory.createElementEntry().intValue("WorkerEventPoolLimit", 1000));
innerElementList.add(EmaFactory.createElementEntry().intValue("TunnelStreamMsgEventPoolLimit",
    2500));
innerElementList.add(EmaFactory.createElementEntry().intValue("TunnelStreamStatusEventPoolLimit",
    3000));
outermostMap.add(EmaFactory.createMapEntry().keyAscii( "GlobalConfig", MapEntry.MapAction.ADD,
    innerElementList ));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().ascii("DefaultIPProvider", "Provider_1"));

innerElementList.add(EmaFactory.createElementEntry().ascii("Server", "Server_1"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Directory", "Directory_1"));

innerElementList.add(EmaFactory.createElementEntry().intValue("ItemCountHint", 5000));
innerElementList.add(EmaFactory.createElementEntry().intValue("ServiceCountHint", 5000));
innerElementList.add(EmaFactory.createElementEntry().intValue("AcceptDirMessageWithoutMinFilters",
    0));
innerElementList.add(EmaFactory.createElementEntry().intValue("AcceptMessageSameKeyButDiffStream",
    0));
innerElementList.add(EmaFactory.createElementEntry().intValue("RefreshFirstRequired", 1));
innerElementList.add(EmaFactory.createElementEntry().intValue("RequestTimeout", 5000));
innerElementList.add(EmaFactory.createElementEntry().intValue("DispatchTimeoutApiThread", 5656));
innerElementList.add(EmaFactory.createElementEntry().intValue("MaxDispatchCountApiThread", 500));
innerElementList.add(EmaFactory.createElementEntry().intValue("MaxDispatchCountUserThread", 500));
innerElementList.add(EmaFactory.createElementEntry().intValue("XmlTraceToStdout", 0));

innerMap.add(EmaFactory.createMapEntry().keyAscii( "Provider_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map("IPProviderList", innerMap));
innerMap.clear();

outermostMap.add(EmaFactory.createMapEntry().keyAscii( "IPProviderGroup", MapEntry.MapAction.ADD,
    elementList));
elementList.clear();
```



```

innerElementList.add(EmaFactory.createElementEntry().ascii("ServerType", "ServerType::RSSL_SOCKET"));
innerElementList.add(EmaFactory.createElementEntry().ascii("CompressionType",
    "CompressionType::LZ4"));
innerElementList.add(EmaFactory.createElementEntry().intValue("GuaranteedOutputBuffers", 7000));
innerElementList.add(EmaFactory.createElementEntry().intValue("NumInputBuffers", 5000));
innerElementList.add(EmaFactory.createElementEntry().intValue("ConnectionPingTimeout", 70000));
innerElementList.add(EmaFactory.createElementEntry().ascii("Port", "14003"));
innerElementList.add(EmaFactory.createElementEntry().intValue("TcpNodeDelay", 1));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Server_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map("ServerList", innerMap));
innerMap.clear();

outermostMap.add(EmaFactory.createMapEntry().keyAscii( "ServerGroup", MapEntry.MapAction.ADD,
    elementList));
elementList.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("DictionaryType",
    "DictionaryType::FileDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("RdmFieldDictionaryItemName", "RWFFld"));
innerElementList.add(EmaFactory.createElementEntry().ascii("EnumTypeDefItemName", "RWFEnum"));
innerElementList.add(EmaFactory.createElementEntry().ascii("RdmFieldDictionaryFileName", "./
    RDMFieldDictionary"));
innerElementList.add(EmaFactory.createElementEntry().ascii("EnumTypeDefFileName", "./enumtype.def"));
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Dictionary_1", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();

elementList.add(EmaFactory.createElementEntry().map("DictionaryList", innerMap));
innerMap.clear();

outermostMap.add(EmaFactory.createMapEntry().keyAscii( "DictionaryGroup", MapEntry.MapAction.ADD,
    elementList));
elementList.clear();

Map serviceMap = EmaFactory.createMap();
ElementList infoElementList = EmaFactory.createElementList();
ElementList stateElementList = EmaFactory.createElementList();
OmmArray infoArray = EmaFactory.createOmmArray();
Series qosSeries = EmaFactory.createSeries();

infoElementList.add(EmaFactory.createElementEntry().intValue("ServiceId", 1));
infoElementList.add(EmaFactory.createElementEntry().ascii("Vendor", "Vendor"));
infoElementList.add(EmaFactory.createElementEntry().intValue("IsSource", 1));
infoElementList.add(EmaFactory.createElementEntry().intValue("AcceptingConsumerStatus", 1));
infoElementList.add(EmaFactory.createElementEntry().intValue("SupportsQoSRange", 1));
infoElementList.add(EmaFactory.createElementEntry().intValue("SupportsOutOfBandSnapshots", 1));
infoElementList.add(EmaFactory.createElementEntry().ascii("ItemList", "#.itemlist"));

```

```

infoArray.add(EmaFactory.createOmmArrayEntry().ascii("MMT_MARKET_PRICE"));
infoArray.add(EmaFactory.createOmmArrayEntry().ascii("MMT_MARKET_BY_PRICE"));
infoArray.add(EmaFactory.createOmmArrayEntry().ascii("MMT_MARKET_BY_ORDER"));
infoArray.add(EmaFactory.createOmmArrayEntry().ascii("130"));
infoElementList.add(EmaFactory.createElementEntry().array("Capabilities", infoArray));
infoArray.clear();

infoArray.add(EmaFactory.createOmmArrayEntry().ascii("Dictionary_1"));
infoElementList.add(EmaFactory.createElementEntry().array("DictionariesProvided", infoArray));
infoArray.clear();
infoArray.add(EmaFactory.createOmmArrayEntry().ascii("Dictionary_1"));
infoElementList.add(EmaFactory.createElementEntry().array("DictionariesUsed", infoArray));
infoArray.clear();

innerElementList.add(EmaFactory.createElementEntry().ascii("Timeliness", "Timeliness::RealTime"));
innerElementList.add(EmaFactory.createElementEntry().ascii("Rate", "Rate::TickByTick"));
qosSeries.add(EmaFactory.createSeriesEntry().elementList(innerElementList));
innerElementList.clear();

infoElementList.add(EmaFactory.createElementEntry().series("QoS", qosSeries));
qosSeries.clear();

stateElementList.add(EmaFactory.createElementEntry().intValue("ServiceState", 1));
stateElementList.add(EmaFactory.createElementEntry().intValue("AcceptingRequests", 1));

innerElementList.add(EmaFactory.createElementEntry().ascii("StreamState", "StreamState::Open"));
innerElementList.add(EmaFactory.createElementEntry().ascii("DataState", "DataState::Suspect"));
innerElementList.add(EmaFactory.createElementEntry().ascii("StatusCode", "StatusCode::DacsDown"));
innerElementList.add(EmaFactory.createElementEntry().ascii("StatusText", "dacsDown"));

stateElementList.add(EmaFactory.createElementEntry().elementList("Status", innerElementList));
innerElementList.clear();

innerElementList.add(EmaFactory.createElementEntry().elementList("InfoFilter", infoElementList));
infoElementList.clear();
innerElementList.add(EmaFactory.createElementEntry().elementList("StateFilter", stateElementList));
stateElementList.clear();

serviceMap.add(EmaFactory.createMapEntry().keyAscii( "DIRECT_FEED", MapEntry.MapAction.ADD,
    innerElementList));
innerElementList.clear();
innerMap.add(EmaFactory.createMapEntry().keyAscii( "Directory_1", MapEntry.MapAction.ADD,
    serviceMap));
serviceMap.clear();

elementList.add(EmaFactory.createElementEntry().map("DirectoryList", innerMap));
innerMap.clear();

outermostMap.add(EmaFactory.createMapEntry().keyAscii( "DirectoryGroup", MapEntry.MapAction.ADD,
    elementList));

```



```
provider = EmaFactory.createOmmProvider(  
    EmaFactory.createOmmIProviderConfig().config(outermostMap).operationModel(  
        OmmIProviderConfig.OperationModel.USER_DISPATCH ), appClient );
```

© 2016 - 2021 Refinitiv. All rights reserved.

Republication or redistribution of Refinitiv content, including by framing or similar means, is prohibited without the prior written consent of Refinitiv. 'Refinitiv' and the Refinitiv logo are registered trademarks and trademarks of Refinitiv.

Any third party names or marks are the trademarks or registered trademarks of the relevant third party.

Document ID: EMAJ362CG.210
Date of issue: June 2021

